ICSEA 2014

The Ninth International Conference on Software Engineering Advances

ISBN: 978-1-61208-367-4

October 12 - 16, 2014

Nice, France

ICSEA 2014 Editors

Herwig Mannaert, University of Antwerp, Belgium

Luigi Lavazza, Università dell'Insubria - Varese, Italy

Roy Oberhauser, Aalen University, Germany

Mira Kajko-Mattsson, Stockholm University & Royal Institute of Technology, Sweden

Michael Gebhart, iteratec GmbH, Germany

ICSEA 2014

Forward

The Ninth International Conference on Software Engineering Advances (ICSEA 2014), held between October 12 - 16, 2014 in Nice, France, continued a series of events covering a broad spectrum of software-related topics.

The conference covered fundamentals on designing, implementing, testing, validating and maintaining various kinds of software. The tracks treated the topics from theory to practice, in terms of methodologies, design, implementation, testing, use cases, tools, and lessons learnt. The conference topics covered classical and advanced methodologies, open source, agile software, as well as software deployment and software economics and education.

The conference had the following tracks:

- Advances in fundamentals for software development
- Agile software techniques
- Improving productivity in research on software engineering
- Software security, privacy, safeness
- Advances in software testing
- Advanced mechanisms for software development
- Specialized software advanced applications
- Software engineering techniques, metrics, and formalisms
- Business technology
- Software deployment and maintenance
- Advanced design tools for developing software
- Software performance
- Web accessibility

Similar to the previous edition, this event continued to be very competitive in its selection process and very well perceived by the international software engineering community. As such, it attracted excellent contributions and active participation from all over the world. We were very pleased to receive a large amount of top quality contributions.

We take here the opportunity to warmly thank all the members of the ICSEA 2014 technical program committee, as well as the numerous reviewers. The creation of such a broad and high quality conference program would not have been possible without their involvement. We also kindly thank all the authors that dedicated much of their time and effort to contribute to ICSEA 2014. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions.

Also, this event could not have been a reality without the support of many individuals, organizations and sponsors. We also gratefully thank the members of the ICSEA 2014 organizing committee for their help in handling the logistics and for their work that made this professional meeting a success.

We hope the ICSEA 2014 was a successful international forum for the exchange of ideas and results between academia and industry and to promote further progress in software engineering research. We also hope that Nice, France provided a pleasant environment during the conference and everyone saved some time to enjoy the charm of the city.

ICSEA 2014 Chairs

ICSEA Advisory Chairs

Herwig Mannaert, University of Antwerp, Belgium
Jon G. Hall, The Open University - Milton Keynes, UK
Mira Kajko-Mattsson, Stockholm University & Royal Institute of Technology, Sweden
Luigi Lavazza, Università dell'Insubria - Varese, Italy
Roy Oberhauser, Aalen University, Germany
Elena Troubitsyna, Åbo Akademi University, Finland
Davide Tosi, Università dell'Insubria - Como, Italy
Luis Fernandez-Sanz, Universidad de Alcala, Spain
Michael Gebhart, iteratec GmbH, Germany

ICSEA 2014 Research Institute Liaison Chairs

Oleksandr Panchenko, Hasso Plattner Institute for Software Systems Engineering - Potsdam, Germany
Teemu Kanstrén, VTT Technical Research Centre of Finland - Oulu, Finland
Osamu Takaki, Gunma University, Japan
Georg Buchgeher, Software Competence Center Hagenberg GmbH, Austria

ICSEA 2014 Industry/Research Chairs

Herman Hartmann, University of Groningen, The Netherlands
Hongyu Pei Breivold, ABB Corporate Research, Sweden

ICSEA 2014 Special Area Chairs

Formal Methods
Paul J. Gibson, Telecom & Management SudParis, France

Testing and Validation
Florian Barth, University of Mannheim, Germany

Web Accessibility
Adriana Martin, National University of Austral Patagonia (UNPA), Argentina

ICSEA 2014 Publicity Chairs

Sébastien Salva, University of Auvergne, Clermont-Ferrand, France

ICSEA 2014

Committee

ICSEA Advisory Chairs

Herwig Mannaert, University of Antwerp, Belgium
Jon G. Hall, The Open University - Milton Keynes, UK
Mira Kajko-Mattsson, Stockholm University & Royal Institute of Technology, Sweden
Luigi Lavazza, Università dell'Insubria - Varese, Italy
Roy Oberhauser, Aalen University, Germany
Elena Troubitsyna, Åbo Akademi University, Finland
Davide Tosi, Università dell'Insubria - Como, Italy
Luis Fernandez-Sanz, Universidad de Alcala, Spain
Michael Gebhart, iteratec GmbH, Germany

ICSEA 2014 Research Institute Liaison Chairs

Oleksandr Panchenko, Hasso Plattner Institute for Software Systems Engineering - Potsdam, Germany
Teemu Kanstrén, VTT Technical Research Centre of Finland - Oulu, Finland
Osamu Takaki, Gunma University, Japan
Georg Buchgeher, Software Competence Center Hagenberg GmbH, Austria

ICSEA 2014 Industry/Research Chairs

Herman Hartmann, University of Groningen, The Netherlands
Hongyu Pei Breivold, ABB Corporate Research, Sweden

ICSEA 2014 Special Area Chairs

Formal Methods
Paul J. Gibson, Telecom & Management SudParis, France

Testing and Validation
Florian Barth, University of Mannheim, Germany

Web Accessibility
Adriana Martin, National University of Austral Patagonia (UNPA), Argentina

ICSEA 2014 Publicity Chairs

Sébastien Salva, University of Auvergne, Clermont-Ferrand, France

ICSEA 2014 Technical Program Committee

Shahliza Abd Halim, Universiti of Technologi Malaysia (UTM) - Skudai, Malaysia
Mohammad Abdallah, Al-Zaytoonah University of Jordan, Jordan
Adla Abdelkader, University of Oran, Algeria
Moataz A. Ahmed, King Fahd University of Petroleum & Minerals – Dhahran, Saudi Arabia
Syed Nadeem Ahsan, TU-Graz, Austria
Mehmet Aksit, University of Twente, Netherlands
Ahmed Al-Moayed, Hochschule Furtwangen University, Germany
Basem Y. Alkazemi, Umm Al-Qura University, Saudi Arabia
Abdullah Alqahtani, University of Dammam, Saudi Arabia / Glasgow Caledonian University, UK
Mohammad Alshayeb, King Fahd University of Petroleum and Minerals, Saudi Arabia
Zakarya A. Alzamil, King Saud University, Saudi Arabia
Vincenzo Ambriola, Università di Pisa, Italy
Jose Andre Dorigan, State University of Maringa, Brazil
Buzzi Andreas, Credit Suisse AG – Zürich, Switzerland
Andreas S. Andreou, Cyprus University of Technology - Limassol, Cyprus
Maria Anjum, Durham University, UK
Paulo Asterio de Castro Guerra, Tapijara Programação de Sistemas Ltda. - Lambari, Brazil
Colin Atkinson, University of Mannheim, Germany
Robert Azarbod, Oracle Corporation, USA
Thomas Baar, Hochschule für Technik und Wirtschaft (HTW) Berlin, Germany
Gilbert Babin, HEC Montréal, Canada
Muneera Bano, International Islamic University - Islamabad, Pakistan
Fernando Sérgio Barbosa, Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco, Portugal
Jorge Barreiros, CITI/UNL: Center of Informatics and Information Technology - UNL || ISEC/IPC: ISEC - Polytechnic Institute of Coimbra, Portugal
Florian Barth, University of Mannheim, Germany
Gabriele Bavota, University of Salerno, Italy
Assia Belbachir, IFSTTAR - Versailles, France
Noureddine Belkhatir, University of Grenoble, France
Simona Bernardi, Centro Universitario de la Defensa / Academia General Militar - Zaragoza, Spain
Jorge Bernardino, Polytechnic Institute of Coimbra - ISEC-CISUC, Portugal
Ateet Bhalla, Oriental Institute of Science and Technology, Bhopal, India
Celestina Bianco, Systelab Technologies - Barcelona, Spain
Christian Bird, University of California, USA
Kenneth Boness, Reading University, UK
Marko Boskovic, Forschungsgesellschaft mbH – Wien, Austria

Samir Ribic, University of Sarajevo, Bosnia and Herzegovina
Elvinia Riccobene, University of Milan, Italy
Daniel Riesco, National University of San Luis, Argentina
Michele Risi, University of Salerno, Italy
Gabriela Robiolo, Universidad Austral, Argentina
Oliveto Rocco, University of Molise, Italy
Rodrigo G. C. Rocha, Federal Rural University of Pernambuco, Brazil
Daniel Rodríguez, University of Alcalá, Madrid, Spain
María Luisa Rodríguez Almendros, Universidad de Granada, Spain
Siegfried Rouvrais, TELECOM Bretagne, France
Mercedes Ruiz Carreira, Universidad de Cádiz, Spain
Mehrdad Saadatmand, Mälardalen University / Alten AB, Sweden
Krzysztof Sacha, Warsaw University of Technology, Poland
Francesca Saglietti, University of Erlangen-Nuremberg, Germany
Sébastien Salva, LIMOS-CNRS / Auvergne University / IUT d'Aubière, France
Maria-Isabel Sanchez-Segura, Carlos III University of Madrid, Spain
Luca Santillo, Agile Metrics, Italy
Gaetana Sapienza, ABB Corporate Research, Sweden
Patrizia Scandurra, University of Bergamo - Dalmine, Italy
Giuseppe Scanniello, Università degli Studi della Basilicata - Potenza, Italy
Klaus Schmid, University of Hildesheim, Germany
Rainer Schmidt, HTW-Aalen, Germany
Christelle Scharff, Pace University, USA
Bran Selic, Malina Software, Canada
Fernando Selleri Silva, Mato Grosso State University (UNEMAT), Brazil
István Siket, University of Szeged, Hungary
Thomas Stocker, University of Freiburg, Germany
Sidra Sultana, National University of Sciences and Technology, Pakistan
Lijian Sun, Chinese Academy of Surveying & Mapping, China
Mahbubur R. Syed, Minnesota State University – Mankato, USA
Davide Taibi, University of Kaiserslautern, Germany
Osamu Takaki, Gunma University, Japan
Giordano Tamburrelli, Università della Svizzera Italiana (USI), Swizterland
Wasif Tanveer, University of Engineering and Technology - Lahore, Pakistan
Nebojša Taušan, University of Oulu, Finland
Pierre Tiako, Langston University, USA
Maarit Tihinen, VTT Technical Research Centre of Finland - Oulu, Finland
Giovanni Toffetti, IBM Research - Haifa, Israel
Maria Tortorella, University of Sannnio - Benevento Italy
Davide Tosi, Università degli studi dell'Insubria - Varese, Italy
Peter Trapp, Ingolstadt, Germany
Elena Troubitsyna, Åbo Akademi University, Finland
Dragos Truscan, Åbo Akademi University, Finland
Mariusz Trzaska, Polish Japanese Institute of Information Technology - Warsaw, Poland

Copyright Information

For your reference, this is the text governing the copyright release for material published by IARIA.

The copyright release is a transfer of publication rights, which allows IARIA and its partners to drive the dissemination of the published material. This allows IARIA to give articles increased visibility via distribution, inclusion in libraries, and arrangements for submission to indexes.

I, the undersigned, declare that the article is original, and that I represent the authors of this article in the copyright release matters. If this work has been done as work-for-hire, I have obtained all necessary clearances to execute a copyright release. I hereby irrevocably transfer exclusive copyright for this material to IARIA. I give IARIA permission or reproduce the work in any media format such as, but not limited to, print, digital, or electronic. I give IARIA permission to distribute the materials without restriction to any institutions or individuals. I give IARIA permission to submit the work for inclusion in article repositories as IARIA sees fit.

I, the undersigned, declare that to the best of my knowledge, the article is does not contain libelous or otherwise unlawful contents or invading the right of privacy or infringing on a proprietary right.

Following the copyright release, any circulated version of the article must bear the copyright notice and any header and footer information that IARIA applies to the published article.

IARIA grants royalty-free permission to the authors to disseminate the work, under the above provisions, for any academic, commercial, or industrial use. IARIA grants royalty-free permission to any individuals or institutions to make the article available electronically, online, or in print.

IARIA acknowledges that rights to any algorithm, process, procedure, apparatus, or articles of manufacture remain with the authors and their employers.

I, the undersigned, understand that IARIA will not be liable, in contract, tort (including, without limitation, negligence), pre-contract or other representations (other than fraudulent misrepresentations) or otherwise in connection with the publication of my work.

Exception to the above is made for work-for-hire performed while employed by the government. In that case, copyright to the material remains with the said government. The rightful owners (authors and government entity) grant unlimited and unrestricted permission to IARIA, IARIA's contractors, and IARIA's partners to further distribute the work.

Table of Contents

Intertwining Relationship Between Requirements, Architecture, and Domain Knowledge

Azadeh Alebrahim, Maritta Heisel

paluno – The Ruhr Institute for Software Technology,
University of Duisburg-Essen, Germany
Email: firstname.lastname@paluno.uni-due.de

***Abstract*—In requirements engineering, properties of the environment and assumptions about it, called *domain knowledge*, need to be captured in addition to exploring the requirements. Despite the recognition of the significance of capturing the required domain knowledge, domain knowledge might be missing, left implicit, or be captured inadequately during the software development process, causing incorrect specifications and software failure. Domain knowledge affects the elicitation and evolution of requirements, the evolution of software architectures, and related design decisions. Conversely, requirements and design decisions affect the elicitation and modification of domain knowledge. In this paper, we propose the iterative capturing and co-developing of domain knowledge with requirements and software architectures. We explicitly discuss the effects of requirements and design decisions on domain knowledge and illustrate this relationship with examples drawn from our research, where we had to go back and forth between requirements, domain knowledge, and design decisions.**

Keywords–quality requirements; requirements engineering; domain knowledge; design decisions; software architecture.

I. INTRODUCTION

It is acknowledged that there is an iterative interplay between requirements and software architecture [1]. Requirements cannot be considered in isolation and should be co-developed with architectural descriptions iteratively, known as *Twin Peaks* as proposed by Nuseibeh [2], to support the creation of sound architectures and correct requirements [3].

The system-to-be comprises the software to be built and its surrounding environment such as people, devices, and existing software [4]. The environment consists of the part of the real world into which the software will be integrated. According to Jackson [5], requirements expressing wishes are properties of the environment that are to be guaranteed by the software (*machine* in Jackson's terminology), whereas there is another class of environment properties that are guaranteed by the environment. Such environment properties and also assumptions about the environment are known as *domain knowledge* [4][6]. Domain properties typically correspond to physical laws. They are invariable, no matter how we build the software. Assumptions have to be satisfied by the environment. They are not guaranteed to be true in every case. For example, when we build a traffic light system to prevent accidents, the assumption is that *drivers stop when they see a red traffic light*. Otherwise, a traffic light system cannot prevent accidents.

Specifications describe the machine and not the environment. Domain knowledge supports the refinement of requirements into implementable specifications [7]. Hence, in requirements engineering domain knowledge needs to be captured in addition to exploring the requirements [4][6].

Despite the recognition of the significance of capturing the required domain knowledge, it might be missing, left implicit, or be captured inadequately during the software development process [4]. Several requirements engineering methods exist, e.g., for security. Fabian et al. [8] concluded in their survey about these methods that there is no state of the art considering domain knowledge yet. Hooks and Farry report on a project in which 49% of requirements errors were due to incorrect domain knowledge [9]. In Colombia in December 1995, capturing inadequate assumptions about the environment of the flight guidance software led to the crash of a Boeing 757 [10].

As software systems become larger and more complex, explicitly capturing domain knowledge becomes crucial. The consideration of domain knowledge is particularly essential when talking about quality requirements since quality requirements such as performance and security rely on specific constraints and assumptions for their satisfaction. For instance, performance is concerned with available *resources* (such as CPU and memory) to process the *workload* [11]. Such resources employed by the software-to-be have specific characteristics such as speed and size that might constrain the satisfaction of quality requirements. Hence, one must explicitly define under which constraints and assumptions a quality requirement will be fulfilled.

We are convinced that during the software development process, domain knowledge is not only used in requirements engineering for obtaining adequate specifications, it also has to be captured during the design phase when selecting patterns and mechanisms or when making design decisions. There are new assumptions and requirements associated with each pattern and quality-specific solution, which have to be considered when deciding on this solution. For example, selecting *asymmetric encryption* as a solution for a confidentiality requirement demands other assumptions regarding the keys and their distribution than *symmetric encryption*.

We distinguish between the domain knowledge related to the problem world, required for obtaining correct specifications, and the domain knowledge which is associated with properties and assumptions about the solution world, required for applying a pattern or mechanism properly. We call the former *Problem-Specific Domain Knowledge (PSDK)*, which is part of the problem peak in the twin peaks model [2] and the latter *Solution-Specific Domain Knowledge (SSDK)*,

 ISBN: 978-1-61208-367-4

as a part of the solution peak. PSDK and SSDK have to be captured for functional requirements and their corresponding functional solutions as well as for quality requirements and their corresponding quality-specific solutions. In this paper, we focus on PSDK and SSDK related to quality requirements and the corresponding solutions.

As an example for the PSDK, we consider the performance. Performance requirements typically describe the time needed for conducting a task (response time). In order to identify and analyze potential performance problems, the *workload* and the available *resources* as performance-relevant domain knowledge have to be captured explicitly (see Section IV-A). As an example for the SSDK, we consider the *symmetric encryption* as a security-specific solution candidate for achieving a confidentiality requirement. This security solution demands new assumptions regarding the *secret key* that have to be elicited explicitly (see Section IV-B).

Capturing SSDK facilitates future design decisions and architecture evolution since the architect knows the consequences of each design decision through necessary assumptions and requirements and can therefore play "what if" scenarios. Consequently, we consider the SSDK as a set of assumptions and facts that builds the foundation for making design decisions. The SSDK represents one input to the design process.

Domain knowledge affects the elicitation and evolution of requirements as well as the evolution of software architectures and related design decisions. Conversely, modification of initial requirements or taking design decisions might lead to capturing new domain knowledge or modifying the existing one. Therefore, apart from the concurrent and iterative development of requirements and architecture, there is an iterative interplay within the problem peak between the requirements and the PSDK and within the solution peak between the design decisions and SSDK. In this paper, we describe these intertwining relationships and propose to co-develop the domain knowledge together with requirements and software architecture. We illustrate these relationships with examples drawn from our research, where we had to go back and forth within each peak and between two peaks.

The contribution of this paper is 1) emphasizing the importance of domain knowledge, particularly quality-relevant domain knowledge and the need for capturing and reusing it in the software development process in a systematic manner, 2) explicitly considering PSDK as part of the problem peak and SSDK as part of the solution peak, and 3) elaborating the intertwining relationship of domain knowledge with requirements and design decisions.

The rest of this paper is structured as follows: In the following, we discuss the related work in Section II. The smart grid scenario as a running example is introduced in Section III. In Section IV, we first present our idea of co-developing requirements, architecture, and domain knowledge. Then, we give examples on how these artifacts affect each other. We conclude the paper in Section V.

II. Related Work

Beside the common and traditional approaches utilizing requirements for creating the software architecture, there have been increasing efforts regarding the intertwining relationship between requirements and architecture. De Boer and van Vliet [12] review different opinions regarding this relationship between requirements as problem description and software architecture as solution description. They propose a closer collaboration between the two communities to profit from the research results that each community provides.

Ferrari and Madhavji [13] conduct an empirical study to analyze the impact of requirements knowledge and experience on the software architecture. Their findings show that architects having knowledge and experience in requirements engineering perform better in terms of architecture quality.

A number of approaches exist that explore the impact of software architecture and design decisions on requirements engineering [14][15][16][17]. Durdik et al. [14] discuss how the results of design decisions can be used to drive requirement elicitation and prioritization. Koziolek [15] proposes to use the feedback from architecture evaluation and design space exploration for prioritizing quality requirements. An exploratory study has been conducted to analyze to what extent an existing software architecture affects requirements engineering [16]. The authors found four types of architectural effects on requirements decisions, namely *enabler*, *constraint*, *influence*, and *neutral*. Woods and Rozanski [17] report on their experience regarding the relationship between the system requirements and software architecture. They propose to use the architecture design to constrain the requirements to a set which is achievable, to frame the requirements, and to inspire new requirements.

Van Lamsweerde [4] and Jackson [6] underline the importance of eliciting domain knowledge in addition to the elicitation of requirements to obtain correct specifications. This corresponds to capturing the PSDK in this paper. Babar et al. [18] emphasize the significance of capturing architecture knowledge for software development. This corresponds to eliciting the SSDK eplicitly.

However, none of these approaches investigate the intertwining relationship between these artifcats. Moreover, to the best of our knowledge, there is no approach exploring the types of effects of requirements, domain knowledge, and design decisions on each other such as capturing new domain knowledge and requirement or modifiying the existing ones.

III. Smart Grid Example

We illustrate our proposed idea through the example of a smart grid system, based on the protection profile that was issued by the Bundesamt für Sicherheit in der Informationstechnik [19]. To use energy in an optimal way, smart grids make it possible to couple the generation, distribution, storage, and consumption of energy. Smart grids use information and communication technology, which allows for financial, informational, and electrical transactions.

Figure 1 shows the simplified context of a smart grid system based on the protection profile [19]. Gateway, the Target Of Evaluation (TOE) is used for collecting, storing, and providing meter data from one or more smart meters which are responsible for one or more commodities, such as electricity, gas, water, or heat. The Local Metrological Network (LMN) represents the in-house data communication netweork which interconnects the smart meters to the gateway. The term Meter refers to a device which is comparable to

 ISBN: 978-1-61208-367-4

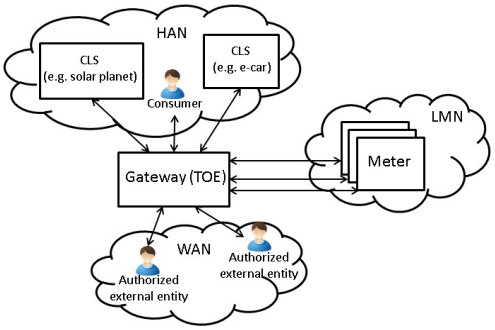


Figure 1. The context of a smart grid system based on [19].

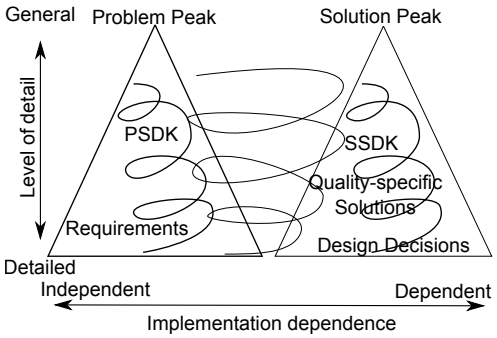


Figure 2. Twin peak model including interrelationships within each peak.

a classical meter with additional functionality. Controllable Local Systems (CLS) are devices of the consumer, such as air condition, solar planet, and intelligent household appliances containing IT-components. They do not belong to the smart metering system. They are in the Home Area Network (HAN) of the consumer. HAN is the in-house data communication netweork interconnecting domestic equipments.

Table I shows an excerpt of terms specific to the smart grid domain taken from the protection profile that are relevant to understand the requirements.

We focus in this paper on the functional requirement *"The smart meter gateway shall submit processed meter data to authorized external entities (RQ4)"*, the security requirements *"Integrity (RQ10)/ Confidentiality (RQ11)/ Authenticity (RQ12) of data transferred in the WAN shall be protected"*, and the performance requirement *"The time to retrieve meter data from the smart meter and publish it through WAN shall be less than 5 seconds (RQ24)"*. We derived these requirements from the protection profile [19] and the report "Requirements of AMI" [20].

IV. Interplay of Requirements, Architecture, and Domain Knowledge

In this section, we describe how requirements, architecture, and domain knowledge affect each other. Figure 2 shows the twin peaks model. It addresses the problem in some linear software development approaches in which on the one hand requirements are elicited, analyzed, and specified in isolation without considering the impact of architecture artifacts. On the other hand design decisions are made without managing the conflicts and making necessary changes in the requirements. The twin peaks model emphasizes the intertwining relationship between requirements and architecture. The spiral between the problem peak and the solution peak in Figure 2 illustrates this relationship between the problem world and the solution world [2].

TABLE I. AN EXCERPT OF RELEVANT TERMS FOR THE SMART GRID

Gateway	represents the central communication unit in a *smart metering system*. It is responsible for collecting, processing, storing, and communicating *meter data*.
Meter data	refers to meter readings measured by the meter regarding consumption or production of a certain commodity.
Smart meter	represents the device that measures the consumption or production of a certain commodity and sends it to the gateway.
Authorized external entity	could be a human or IT unit that communicates with the gateway from outside the gateway boundaries through a *Wide Area Network (WAN)*.
WAN	WAN provides the communication network that interconnects the gateway with the outside world.

In this paper, we emphasize the need to capture and specify requirements and PSDK as it affects the elicitation and evolution of requirements. Equally, evolving requirements might have an affect on previously captured PSDK. The spiral in the first peak, the problem peak in Figure 2 represents the synergistic relationship between the requirements and the PSDK. Similarly, architecture artifacts and SSDK exhibit such intertwining relationship as design decisions and trade-offs might require the elicitation of new SSDK or modification of the existing one. Figure 2 shows the relationship between the design decisions and the SSDK by means of the spiral in the solution peak.

In the following, we describe the impact of requirements and PSDK on each other in the problem peak. Then, the impact of design decisions and SSDK on each other is described. We give examples of such effects using the smart grid example.

A. Interplay of Requirements and PSDK

For developing software that achieves its desired quality requirements, additional information (PSDK) for each quality requirement must be explicitly elicited. As an example, we consider the performance. As mentioned in Section I, the *workload* and the available *resources* as performance-relevant domain knowledge have to be elicited and incorporated into existing requirement models explicitly. The workload is described by triggers of the system, representing requests from outside or inside the system. Workload exhibits the characteristics of the system use. It includes the number of concurrent users and their arrival pattern. The arrival pattern can be periodic (e.g., every 10 milliseconds), stochastic (according to a probabilistic distribution), or sporadic (not to capture by periodic or stochastic characterization) [21]. Processing the requests requires resources. Each resource has to be described by its type in the system, such as CPU, memory, and network, its utilization, and its capacity, such as the transmission speed for a network.

Performance-relevant domain knowledge can be gained from performance experts and analysts. In this paper, we do not aim at proposing approaches on how to elicit and model performance-relevant domain knowledge, but at emphasizing the need for eliciting domain knowledge as additional information to the quality requirements and annotating it properly

 ISBN: 978-1-61208-367-4

TABLE II. INSTANTIATED PSDK TEMPLATE FOR *RQ24* (AND MAPPING TO THE MARTE PROFILE)

Quality: Performance, Quality Requirement: RQ24					
PSDK Template					**Mapping to MARTE**
PSDK Description		**Possible Values**	**Value**	**Range of Value**	**Property**
Number of concurrent users		Natural	50		GaWorkloadEvent. pattern. population
Arrival pattern		ArrivalPattern	closed		GaWorkloadEvent. pattern
Data size		DataSize (bit, Byte, KB, MB, GB)	640 MB		GaStep. msgSize
Memory	capacity	DataSize (bit, Byte, KB, MB, GB)	-		HwMemory. memorySize
	latency	Duration (s, ms, min,hr, day)	-		HwMemory. timing
Network	bandwidth	DataRate (b/s, Kb/s, Mb/s)	2.4 Kb/s		HwMedia. bandWidth
	latency	Duration (s, ms, min,hr, day)	-		HwMedia. packetTime
CPU	speed	Frequency (Hz, kHz, MHz, GHz)	470 MHz		HwProcessor. frequency
	Number of cores	Natural	1		HwProcessor. nbCores

in the requirement models as initially proposed in our previous work [22]. We propose to document the PSDK for the corresponding software qualities as structured templates. We call such templates *PSDK Templates* to be instantiated separately for each type of software quality. The instantiated template should be known to the requirements engineer or performance analyst to analyze whether a particular performance requirement can be satisfied or not. In order for the analyst to be able to determine "whether the meter data can be transferred through WAN within 5 seconds", (s)he needs PSDK involving the number of concurrent users in the system, the bandwidth of the network, the CPU speed and the core numbers, and the data volume which is transferred over the network. We exemplify the instantiation of the PSDK template for the performance requirement *RQ24* according to the information contained in the existing documents for the smart grid application [19][20]. Table II shows the instantiated PSDK template (see column *Value*). The columns *PSDK Description* and *Possible Values* show the domain knowledge to be elicited for performance and its possible values. We may extend the template with the column *Range of Value* showing the possible range of values. Such information can be obtained from documents or involved stakeholders. We make use of this column later on to modify (strengthen or relax) the PSDK as a conflict resolution strategy.

In addition to the template, a suitable notation for integrating quality-relevant domain knowledge in the requirement models can be selected to be used for requirements analysis in a model-based approach. We select the UML profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) [23] adopted by OMG consortium that allows us to annotate performance-relevant domain knowledge in a UML-based modeling approach. The column *Property* shows the corresponding stereotypes from the MARTE profile.

We describe the need for eliciting PSDK and documenting it in PSDK templates when we elicit quality requirements by the example of a performance requirement. Requirements might affect the PSDK when they are modified for any reason. In such a case, the PSDK has to be checked for possible modifications.

One important source for changes in the requirements is detecting and resolving conflicts among requirements. Such changes might cause changes in the PSDK [24]. Requirements conflicts can typically be resolved on the requirement level by relaxing the conflicting requirements or on the architecture level by relaxing the corresponding solutions. In some cases, only potential conflicts among requirements can be detected and not the genuine ones [25]. Hence, in such cases the conflict resolution shall preferably be postponed to the architecture level, where more details are available for detecting the genuine requirement conflicts and resolving them. Resolving requirements conflicts by relaxing requirements might lead to the modification of the existing PSDK (e.g., assumptions). Resolution of conflicting requirements on the requirement level is one source for updating existing domain knowledge. One might make a trade-off between a performance requirement and a security requirement by relaxing the performance requirement.

In our previous work [24], we proposed a method for detecting and resolving conflicts among performance and security requirements, as security has mostly a deep impact on the performance of the whole system. The reason is that mechanisms and patterns for satisfying security requirements, such as encryption or Message Authentication Code (MAC) are time-consuming. The general principle of our method for detecting interactions among requirements is using the structure of requirement models to identify trade-off points, where security and performance requirements might interact. After we have identified pairs of conflicting requirements, we have made trade-offs by relaxing one or both conflicting requirements to resolve the conflict. As an example, we consider two requirements *RQ11* and *RQ24* that we identified as conflicting. One option for relaxing the performance requirement is modifying the performance-relevant domain knowledge. Typically, domain knowledge consists of facts (domain properties in [4]) that we cannot change, relax, and negotiate and assumptions that can be changed, relaxed, and negotiated [4]. For resolving the conflict, we might modify (relax or strengthen) the assumptions. For example, the *number of concurrent users* is not a fixed property. It is an assumption and can be modified, when the performance requirement cannot be achieved with this assumption. Hence, we modify *number of concurrent users* by reducing it to a number less than 50. The same holds true for *data size*, which has to be reduced to less than 640 MB, *network bandwidth*, which has to be relaxed to more than 2.4 Kb/s (see the instantiated PSDK in Table II). The rest of the properties are either fixed (can be considered as facts) or irrelevant for the corresponding requirement, or unknown and thus cannot be considered for the modification process. We document such modifications in the column *Range of Value*. Note that changes in the requirements or PSDK should be negotiated with the stakeholders.

B. Interplay of Design Decisions and SSDK

Quality-specific patterns and mechanisms for performance such as *load balancer* and *master worker* [26] and for security such as *Role-Based Access Control (RBAC)* and *encryption* [27] represent solution candidates for achieving quality

TABLE III. IMPACT OF SECURITY-SPECIFIC SOLUTION *ASYMMETRIC ENCRYPTION* ON REQUIREMENTS AND SSDK

Security-specific Solution	
Name	Asymmetric Encryption
Brief Description	The plaintext is encrypted using the public key and decrypted using the private key.
Quality Requirement to be achieved	Security (confidentiality)
Positively affected quality requirement(s)	-
Negatively affected quality requirement(s)	Performance
Necessary Conditions	
Quality Requirement □ SSDK □	Integrity of public key during transmission shall be/is preserved.
Quality Requirement □ SSDK □	Confidentiality of private key during transmission shall be/is preserved.
Quality Requirement □ SSDK □	Integrity of private key during transmission shall be/is preserved.
Quality Requirement □ SSDK □	Confidentiality of private key during storage shall be/is preserved.
Quality Requirement □ SSDK □	Integrity of private key during storage shall be/is preserved.
Quality Requirement □ SSDK □	Integrity of public key during storage shall be/is preserved.

TABLE IV. IMPACT OF SECURITY-SPECIFIC SOLUTION *SYMMETRIC ENCRYPTION* ON REQUIREMENTS AND SSDK

Security-specific Solution	
Name	Symmetric Encryption
Brief Description	The plaintext is encrypted and decrypted using the same secret key.
Quality Requirement to be achieved	Security (confidentiality)
Positively affected quality requirement(s)	-
Negatively affected quality requirement(s)	Performance
Necessary Conditions	
Functional Requirement □ SSDK □	Secret key shall be/is distributed.
Quality Requirement □ SSDK □	Confidentiality and integrity of secret key distribution distribution shall be/is preserved.
Quality Requirement □ SSDK □	Confidentiality of secret key during transmission shall be/is preserved
Quality Requirement □ SSDK □	Confidentiality of secret key during storage shall be/is preserved.
Quality Requirement □ SSDK □	Integrity of secret key during storage shall be/is preserved.
Quality Requirement □ SSDK □	Confidentiality of encryption machine shall be/is preserved.
Quality Requirement □ SSDK □	Integrity of encryption machine shall be/is preserved.
Quality Requirement □ SSDK □	Confidentiality of plaintext shall be/is preserved.

requirements. By exploring the solution space for achieving quality requirements, we require to know which assumptions and facts are to be considered, and which new functional and quality requirements are to be elicited when deciding on a particular solution. Generally speaking, all information that can affect the requirements and related domain knowledge has to be documented.

We propose to provide a template consisting of two parts for the analysis of quality-specific solutions and their impacts on requirements and domain knowledge. We analyze the impact of the solution candidate on the problem space (i.e., requirements), and on the SSDK. Such a template helps us later on when selecting a particular solution candidate to keep track on changes in the requirements and domain knowledge. It also supports an unexperienced architect in understanding the impact of design decisions on the entire system, particularly on the achievement of quality requirements.

The first part of the template (see Table III) accommodates information about the quality-specific solution itself, such as name (*Name*), description (*Brief Description*), the quality requirement which will be achieved when selecting this solution (*Quality Requirement to be achieved*), and the quality requirements which are positively or negatively affected by this solution ((*Positively affected Quality Requirement*), (*Negatively affected Quality Requirement*)). For example, improving the security may result in decreasing the performance. Hence, the impact of each security-specific solution on other quality requirements has to be captured in the first part of the template.

The second part of the template captures and documents necessary conditions which have to be addressed when selecting this solution. Necessary conditions have to be addressed either as *Functional Requirement*, *Quality Requirement*, or as *SSDK*. We elicit the necessary conditions as requirements if the software to be built shall achieve them. In contrast, assumptions have to be satisfied by the environment [4]. Hence, the necessary conditions have to be captured as assumptions (one part of the SSDK) if they have to be satisfied by the environment. Assumptions are not guaranteed to be true in every case. For the case that we assume the environment (not the machine) takes the responsibility for meeting them, we capture them as assumptions. This should be negotiated with the stakeholders and documented properly. Tables III and IV show such a template for the security-specific solutions *asymmetric encryption* and *symmetric encryption*.

We describe Table IV in more detail. After capturing the basic information about the security-specific solution *symmetric encryption* in the first part, in the second part we elicit new requirements and capture new assumptions that arise with the solution, such as *secret key shall be/ is distributed*. Eliciting this condition results in thinking about security issues concerned with it, such as *confidentiality and integrity of secret key distribution shall be/is preserved*. If we require

that the software we build is responsible for preserving the confidentiality and integrity of the secret key not only during the transmission but also during the storage, we have to capture these as requirements. This is the reason why the necessary conditions are presented as checkboxes to be selected by checking the relevant checkbox as requirement or SSDK.

As mentioned before, one option for resolving interactions among requirements is making trade-offs between corresponding quality-specific solutions. To this end, one or both quality-specific solutions have to be relaxed. Making such design decisions requires eliciting or updating SSDK and requirements associated with the particular solution. For example, selecting a *symmetric encryption* for achieving a confidentiality requirement instead of an *asymmetric encryption* demands other assumptions and requirements with respect to the required keys and key distribution as shown in the corresponding Templates (Tables III and IV). To demonstrate this idea, we consider the *asymmetric encryption* as the initial security-specific solution, which is selected for satisfying the security requirement *RQ11*. *RQ11* is concerned with transmitting meter data through the WAN in a confidential way. *Asymmetric encryption* provides sufficient protection during transmitting meter data through the WAN so that the confidentiality requirement can be achieved. However, by applying our method for detecting interactions among quality requirements in our previous work [24], we detected a conflict with the performance requirement *RQ24*. Hence, *RQ24* cannot be achieved in less than 5 seconds when keeping the security-specific solution *asymmetric encryption* for meeting the security requirement *RQ11*. We have to decide for a strategy to resolve the conflict. We can relax the performance requirement by increasing the response time as one possible resolution strategy. Strengthening or relaxing the PSDK for example by raising the *network bandwidth* or by decreasing the *data size* is possible as well, as described in Section IV-A. Such strategies are at the cost of the performance requirement *RQ24* and can only be used if the security requirement *RQ11* has a higher priority. Here, we assume that the performance requirement *RQ24* has a higher priority. Hence, we have to make a trade-off by relaxing the security-specific solution. This can be achieved by selecting another security-specific solution, which is faster. We decide on *symmetric encryption* instead of *asymmetric encryption*. Symmetric encryption is faster than the asymmetric encryption. It, however, demands other requirements and SSDK. In contrast to the asymmetric encryption, which uses different keys for encrypting and decrypting data, the symmetric encryption uses only one key. Thus, we have to care about the key distribution. Hence, this design decision leads to changes in the requirements as well as in the domain knowledge as shown in Table IV.

V. Conclusion and Future Work

In this paper, we underlined the importance of capturing and documenting domain knowledge, particularly quality-relevant domain knowledge for the problem space as well as for the solution space. More importantly, we described how requirements, design decisions and domain knowledge affect each other. Eliciting and updating requirements causes elicitation and modification of PSDK. Capturing and evolving solutions on the architecture level requires eliciting and modifying SSDK and requirements. Hence, domain knowledge should be captured and developed iteratively and incrementally with requirements and architecture to achieve adequate specifications.

In order to be able to argue that the requirements will be satisfied under specific constraints and assumptions, PSDK should be traceable to the requirements [4]. Moreover, design decisions should be traceable to SSDK and requirements to reflect the changes in design decisions and software architecture to the problem peak. Keeping the changes consistent in requirements, domain knowledge (PSDK and SSDK), and software architecture is challenging. Model-based approaches enables us to provide support by keeping such trace information in the model. In the future, we want to provide traceability links between these artifacts in our models to keep track of the changes emerging in one peak which cause changes in the other peak.

Acknowledgments.

This research was partially supported by the German Research Foundation (DFG) under grant numbers HE3322/4-2. We thank Thein Than Tun for his useful feedback on our work.

References

[1] M. Mirakhorli and J. Cleland-Huang, “Traversing the twin peaks,” IEEE Software, vol. 30, no. 2, 2013, pp. 30–36.

[2] B. Nuseibeh, “Weaving together requirements and architectures,” IEEE Computer, vol. 34, no. 3, 2001, pp. 115–117.

[3] M. Whalen, A. Gacek, D. Cofer, A. Murugesan, M. Heimdahl, and S. Rayadurgam, “Your "What" Is My "How": Iteration and Hierarchy in System Design,” IEEE Software, vol. 30, no. 2, 2013, pp. 54–60.

[4] A. Lamsweerde, Requirements Engineering: From System Goals to UML Models to Software Specifications. Wiley, 2009.

[5] M. Jackson, “The meaning of requirements,” Ann. Softw. Eng., vol. 3, Jan. 1997, pp. 5–21.

[6] ——, Problem Frames. Analyzing and structuring software development problems. Addison-Wesley, 2001.

[7] P. Zave and M. Jackson, “Four dark corners of requirements engineering,” ACM Trans. Softw. Eng. Methodol., vol. 6, 1997, pp. 1–30.

[8] B. Fabian, S. Gürses, M. Heisel, T. Santen, and S. Schmidt, “A comparison of security requirements engineering methods,” Requirements Engineering – Special Issue on Security Requirements Engineering, vol. 15, 2010, pp. 7–40.

[9] I. F. Hooks and K. A. F., Customer-centered Products: Creating Successful Products Through Smart Requirements Management. AMACOM, 2001.

[10] F. Modugno, N. Leveson, J. Reese, K. Partridge, and S. Sandys, “Integrated safety analysis of requirements specifications,” Requirements Engineering, 1997, pp. 65–78.

[11] L. Bass, M. Klein, and F. Bachmann, “Quality attributes design primitives,” Software Engineering Institute, Tech. Rep., 2000.

[12] R. C. de Boer and H. van Vliet, “Controversy corner: On the similarity between requirements and architecture,” J. Syst. Softw., vol. 82, no. 3, Mar. 2009, pp. 544–550.

[13] R. Ferrari and N. H. Madhavji, “The Impact of Requirements Knowledge and Experience on Software Architecting: An Empirical Study,” in Proceedings of the 6th Working IEEE/IFIP Conference on Software Architecture (WICSA), 2007, p. 16.

[14] Z. Durdik, A. Koziolek, and R. Reussner, “How the understanding of the effects of design decisions informs requirements engineering,” in 2nd International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks), 2013, pp. 14–18.

[15] A. Koziolek, “Research preview: Prioritizing quality requirements based on software architecture evaluation feedback,” in Requirements Engineering: Foundation for Software Quality. Springer, 2012, vol. LNCS7195, pp. 52–58.

 ISBN: 978-1-61208-367-4

[16] J. Miller, R. Ferrari, and N. Madhavji, “Architectural effects on requirements decisions: An exploratory study,” in Proceedings of the 7th Working IEEE/IFIP Conference on Software Architecture (WICSA), 2008, pp. 231–240.

[17] E. Woods and N. Rozanski, “How Software Architecture can Frame, Constrain and Inspire System Requirements.” in Relating Software Requirements and Architectures. Springer, 2011, pp. 333–352.

[18] M. Ali Babar, I. Gorton, and R. Jeffery, “Capturing and Using Software Architecture Knowledge for Architecture-Based Software Development,” in Proceedings of the 5th International Conference on Quality Software, ser. QSIC ’05. IEEE Computer Society, 2005, pp. 169–176.

[19] H. Kreutzmann, S. Vollmer, N. Tekampe, and A. Abromeit, “Protection profile for the gateway of a smart metering system,” BSI, Tech. Rep., 2011.

[20] Remero et al., “D1.1 Requ. of AMI,” OPEN meter proj., Tech. Rep., 2009.

[21] L. Bass, P. Clemens, and R. Kazman, Software architecture in practice. Addison-Wesley, 2003.

[22] A. Alebrahim, M. Heisel, and R. Meis, “A structured approach for eliciting, modeling, and using quality-related domain knowledge,” in Proceedings of the 14th International Conference on Computational Science and Its Applications (ICCSA), ser. LNCS. Springer, 2014, vol. 8583, pp. 370–386.

[23] UML Revision Task Force, UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems, 2011, http://www.omg.org/spec/MARTE/1.0/PDF [retrieved: 2014-08-07].

[24] A. Alebrahim, C. Choppy, S. Faßbender, and M. Heisel, “Optimizing functional and quality requirements according to stakeholders’ goals,” in System Quality and Software Architecture (SQSA). Elsevier, 2014, pp. 75–120.

[25] A. Alebrahim, S. Faßbender, M. Heisel, and R. Meis, “Problem-Based Requirements Interaction Analysis,” in Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ), ser. LNCS, vol. 8396. Springer, 2014, pp. 200–215.

[26] C. Ford, I. Gileadi, S. Purba, and M. Moerman, Patterns for Performance and Operability. Auerbach Publications, 2008.

[27] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad, Security patterns: integrating security and systems engineering. John Wiley & Sons, 2005.

 ISBN: 978-1-61208-367-4

Unified Conceptual Model for Joinpoints in Distributed Transactions

Anas M. R. AlSobeh, Stephen W. Clyde
Computer Science Department
Utah State University
Logan, Utah, USA
aalsobeh@aggiemail.usu.edu, Stephen.Clyde@usu.edu

***Abstract*—Distributed transaction processing systems can be unnecessarily complex when crosscutting concerns, e.g., logging, concurrency controls, transaction management, and access controls, are scattered throughout the transaction processing logic or tangled into otherwise cohesive modules. Aspect orientation has the potential of reducing this kind of complexity; however, currently, aspect-oriented programming languages and frameworks only allow weaving of advice into contexts derived from traditional executable structures. This paper lays a foundation for weaving advice into distributed transactions, which are high-level runtime abstractions. To establish this foundation, we capture key transaction events and context information in a conceptual model, called *Unified Model for Joinpoints Distributed Transactions* (UMJDT). This model defines interesting joinpoints relative to transaction execution and context data for woven advice. A brief discussion of advice weaving and the potential for reducing complexity with transaction-specific aspects is provided, but the details of the actual weaving are left for another paper. Also, this paper suggest further research for studying the modularity and reuse achieved through the ability to weave crosscutting concern into transaction directly.**

Keywords-complexity; modularity; distributed transaction; joinpoint; operation; context; advice; aspect; crosscutting concerns.

I. INTRODUCTION

Frederick Brooks characterizes software complexity as either *essential* or *accidental*, where essential complexity stems from the very nature of the problem being solved by the software and accidental complexity comes from the way that the problem is being solved [1]. A *Distributed Transaction Processing System* (DTPS) may have essential complexity in the nature of the data, operations on the data, or the volume of data. However, issues such as logging, persistence, resource location, and even distribution itself are more likely to be sources of accidental complexity, because they are not usually inherent parts of the problem. When these issues are secondary to the primary purposes of a DTPS, it is common to find logic for them scattered throughout the software and tangled into core application logic. For example, concurrency-control operations, like locking and unlocking, may be spread throughout the system and be implemented with similar snippets of code.

Aspect Orientation (AO), an extension to *Object Orientation* (OO), can help manage both essential and accidental complexity by localizing and encapsulating crosscutting concerns in first-class software components, called *aspects* [2]. An aspect is very much like a class in OO and an aspect instance is like an object, except that an aspect defines special methods, called *advices*, which are automatically woven into the core application according to specifications, called *pointcuts*. However, existing AO Programming Languages (AOPLs) and frameworks only allow the weaving of advice into the execution of code-based contexts, such as methods, constructors, and exceptions. They do not directly allow behaviors to be woven into more abstract contexts, such as transactions.

One could argue that a good programmer can do the same thing in OO by defining classes for the crosscutting concerns and hard coding calls to methods of those classes in all the right places. However, the issue is not whether it can be done; rather, it is the difference in abstractions. AO offers better abstractions for separating crosscutting concerns from core functionality that do require core functionality to dependent on crosscutting concerns in any way. An AO developer should be able to add/remove aspects to/from a project without changes to any other code. Some authors refer to this as a principle, called *obliviousness* [3].

A transaction is a set of operations on shared resources, such that its execution results in either the successful completion of all operations or the completion of no operation. Besides this all-or-nothing property, called *atomicity*, transactions are *consistent*, *isolated*, and *durable*, meaning that persistent data will only change from one valid state to another, other concurrent transactions cannot see the effects of a transaction until it completes, and that effects of a transaction become persistent after completion even if there is system failure. Together, *atomicity*, *consistency*, *isolation*, and *durability* are often referred to as the ACID properties [3][5].

Distributed transactions are transactions, but their operations are executed on multiple host machines, ideally with improved throughput. From a logical perspective, a distributed transaction can be a flat sequence of operations or a hierarchy of sub-transactions, also known as *nested transactions*. In the latter case, nested transactions may execute concurrently and still observe the ACID properties.

Regardless of whether a distributed transaction is a flat sequence of operations or comprised of nested transactions, it is an ephemeral concept that spans multiple execution threads and operations using distributed resources. Therefore, from an execution perspective, it may seem non-contiguous and unevenly spread over time and space. A transaction's context is not tied to code constructs, like constructors and

 ISBN: 978-1-61208-367-4

methods, in a single thread of execution; rather, it consists of loosely-coupled abstractions like dynamically generated identifiers, timestamps, and tentative value sets for distributed resources. This makes its very difficult for AO developers to localize and encapsulate crosscutting concerns that apply to transactions as execution units.

This paper takes a preliminary step in enabling AO developers to treat transactions as first-class concepts into which compilers or frameworks can weave crossing concerns. Specifically, it unifies DTPS concepts related to a) transactions in general, b) the kinds of information that comprise their context, and c) events that represent interesting time points/places for when/where the crosscutting concerns might augment an application's core functional or the underlying transaction processing system.

Section II provides more detail about aspect-oriented programming concepts and background about common transaction concepts. Section III proposes possible joinpoints in the execution of distributed transactions and relevant the context information for each. Section IV presents a sample of transaction-related crosscutting concerns. Section V presents the UMJDT model and discusses two key areas, namely a transaction context and joinpoints. Although the technical details of advice weaving are beyond the scope of this paper, Section VI provides an outline of the process and highlights some of the key issues. Section VII summarizes the contributions on this paper and discusses next steps.

II. BACKGROUND

A. Overview of Aspect Orientation

As mentioned above, AO is an extension to OO that allows developers to extract and untangle secondary concerns from the primary features of an application. It is difficult to define what constitutes a secondary concern in general because it depends on the purpose of the software being built. However, secondary concerns often show up in less-than-expertly-designed OO software as similar snippets of code scattered across multiple modules or tangled into methods that primarily serve other purposes. A common example is tracing or logging in a data processing application, where the developers want a chronology of the execution for either system verification, audit-trail, or performance-monitoring measurement reasons. To do this, they might insert logic throughout the code that writes various messages or statistics to a file. Eventually, these log-writing code snippets become scattered across the software and tangled in otherwise cohesive methods.

An AOPL, like AspectJ [6], would allow a developer to remove all of the log-writing code from the main application and place that logic in an *aspect*, which is a class-like abstract data type. An aspect can include data members, methods, nested types and everything else a class can include. However, they can also include *advices* and *pointcuts*. An advice is like a method because it implements some specific behavior; however, it is not invoked like a method. Instead, the AOPL's compiler or runtime environment *weaves* the advice into the system so it is executed at specific places and time defined by *pointcuts*. A *pointcut* is a pattern that identifies a set of *joinpoints*, which are best characterized as intervals within program's execution flow. Examples of joinpoints in typical AOPL's include the execution of a method or the setting of a property. Consequently, their start and end points map to specific elements of the code, called *shadows*, which correspond to places where those intervals may start or end. The weaving of advice into the shadows is an automated process, and understanding it in depth is not necessary to appreciate the contributions of this paper. We refer readers interested in learning more about weaving of advice to the overview of AspectJ by Kiczales, et al. [6].

When advice executes, it can access context information about the joinpoint at which it was invoked. This context includes the location of the joinpoint (i.e., the shadow) and runtime information about the objects involved. Some on the context information is static and therefore can be computed during weaving; other context is dynamic and depends on the objects involved in the joinpoint.

B. Transaction Concepts

As mentioned, the objective of this paper is to lay the foundation for weaving crosscutting concerns into transactions in DTPS's. This requires identifying the logical places, i.e., joinpoints, in transaction execution where a developer might want to weave advice, as well as the kinds of information that should be available in joinpoint context.

There are many different DTPS's in use today and they vary in terms of features and implementations. However, they share commonalities in their underlying concepts of transaction distribution, management, execution, and concurrency control. It is on these basic concepts that we will focus our attention and lay a foundation for identifying transaction joinpoints and context.

As with transactions in centralized systems, a distributed transaction is a sequence of operations on shared resources that observe the ACID properties [7][8]. The difference is that the operations of a distributed transaction execute on more than one host machine, which opens up the possibility of subsequences of those operations executing concurrently, without shared memory to help with concurrency controls.

In general, a distributed transaction can be thought of as a tree of operations, instead of strict sequence. To visualize this, consider a simple example of a transaction-based manufacturing system that builds *Widgets* from *Goo* and *Gadgets* from *Widgets*. See Figure 1. The *Goo*, *Widget*, and *Gadgets* are all stored in "piles". The individual objects and the piles of objects are all shared resources. This system also includes processing components, i.e., shared resources, that handle the manufacturing. Specifically, there are *Builders* that create *Raw Widgets* from *Goo*, *Bakers* that turn *Raw Widgets* into *Rough Widgets* and *Polishers* that refine *Rough Widgets* into *Polished Widgets*. Finally, there are *Assemblers* that create *Gadgets* from *Widgets* and *Labelers* that tag the

 ISBN: 978-1-61208-367-4

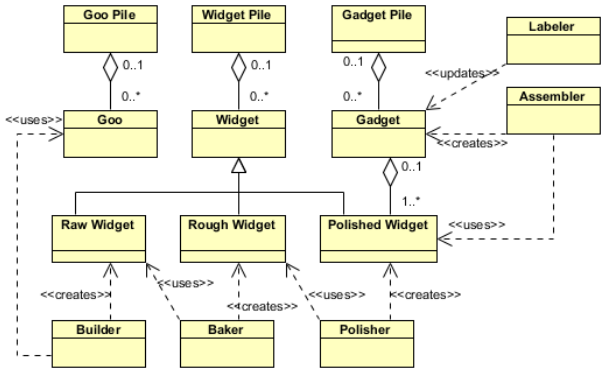


Figure 1 - Resources in a Widget and Gadget Manufacturing System.

a) Transaction T1
Op1.1: Get Goo from Goo Pile
Op1.2: Give Goo to a Builder and get back a Raw Widget
Op1.3: Give Raw Widget to a Baker and get a Rough Widget
Op1.4: Give Rough Widget to a Polisher and get a Polished Widget
Op1.5: Put Polish Widget in a Widget Pile

b) Transaction T2
Op2.1: Get Widget (W1) from Widget Pile 1
Op2.2: Get Widget (W2) from Widget Pile 2
Op2.3: Give W1 and W2 to Assembler and get a Gadget, G
Op2.4: Put Gadget G in a Gadget Pile
Op2.5: Have Labeler put a tag on G

Figure 2 - Two Sample Transactions for Constructing Widgets and Gadgets.

Gadgets with serial numbers. Figure 2 lists two simple transactions that represent a) the construction of a *Polished Widget* and b) the construction of a *Gadget* from two *Widgets.*

Now assume that piles of *Goo*, *Widgets*, and *Gadgets* are distributed across many locations (hosts) and that *Builders* are at the same location as *Goo Piles*; *Bakers* and *Polishers* are at the same location as *Widget Piles*; and *Assemblers* and *Labelers* are close to *Gadget Piles*, but not necessarily at the same location. With this distribution of resources, transaction *T2* could execute in a distributed manner by having *Op2.1* execute in a sub-transaction, *ST2.1*, *Op2.2* execute in another sub-transaction, *ST2.2*, both on the same host as the desired *Widget Pile*, and *Op2.3*-*Op2.5* in a sub-transaction, *ST2.3*, on the same host as the desired *Gadget Pile*. Figure 3 represents this distributed transaction as a simple tree with *T2* as the root and the operations as the leaves.

T1 and *T2* are just two concrete transactions, but this system could have hundreds of similar transactions running at the same time. As in all DTPS, each transaction receives a unique identity, i.e., *Transaction Identifier* (TID), when it starts. All references to a transaction will be via this identifier. Typically, in a DTPS, a *Transaction Manager* (TM), is responsible for assigning TID's and keeping track of parent/sub-transactions relationships.

Beside TID assignment, TM's are also typically responsible for starting transactions (and sub-transactions), and ending transactions by either committing or aborting the results. A TM may also oversee the execution of transaction

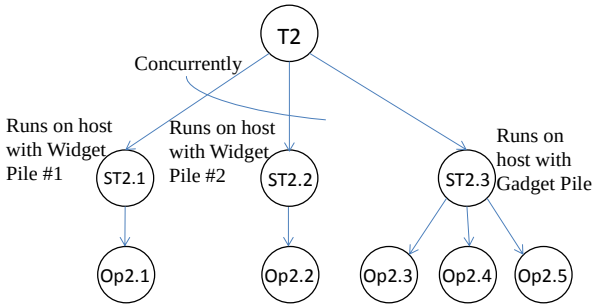


Figure 3 - Possible Distribution of Transaction T2.

operations on resources and any necessary concurrency controls, such as locking, for those resources. Some DTPS delegate these responsibilities to separate components such as *Resource Managers* and *Lock Managers*, but such architectural differences are not important here. For the purpose of exploring possible transaction-related joinpoints and context information, it is important to just recognize that operation execution and concurrency control take place with respect to individual resources.

Finally, a TM can also track information about its execution environment, including information about threads of execution, processes, host machines, secondary storage, and even network connections. It may do this for a variety of reasons, including performance management, audit trails, and recovery in case of failure.

A transaction is typically broken up into two basic phases: an execution phase and a commit phase [8]. The execution phase is considered tentative, because the changes are not made permanent until the commit phase. During the execution phase, the TM performs the operations in the body within its own context. Logically, the operations may result in the tentative changes to shared resources. In a commit phase, the TM will either finalize all of the tentative changes or abort the transaction.

Three common approaches to concurrency controls are *optimistic*, *timestamp-based*, and *pessimistic. Optimistic* approaches to concurrency control allow conflicts to occur during the tentative phases of concurrent transactions, then leave it up to the TM to detect conflicts and abort one or more transactions when they occur, using either *forward or backward validation* [9][10]. *Timestamp-based* approaches guarantee *serial equivalence* [11] by imposing an ordering on the execution of the operations in the tentative phase. *Pessimistic* approaches use locks to prevent conflicts from occurring in the tentative phase of execution. They do this by delaying operation execution or by trigging an abort (in the case of deadlock [12]). Locking schemes vary, but are all based on premise that a transaction must hold a particular kind of lock before performing an operation.

A common and simple locking scheme consists of two types of locks: one for read operations and one write for operations [12]. The pseudo-code in Figure 4 includes requests for the appropriate read and writes locks, following this simple scheme.

 ISBN: 978-1-61208-367-4

A transaction's context information includes those pieces of data and metadata that the transaction needs to be self-contained, guarantee the ACID properties, and support correct execution of both the tentative and commit phases of execution. Supporting correct execution of the commit phase means that the context needs to include sufficient information for the TM to decide whether the transaction conflicts with other concurrent transactions. However, the details of this context data depend heavily on the implementation of the DTPS, the types of concurrency control in use, and the commit algorithm. The only data that are common to virtually all DTPS are the TID and a reference (direct or indirect) to the responsible TM. Beyond these two items, a transaction's context may include many different kinds of implementation specific data, e.g., sets of tentative values, rollback logs, snapshots, lock information, timestamps, and other kinds of metadata. Therefore, any system that aims to support aspects for transaction must allow for context information to contain data that specific to a DTPS's implementation.

III. POTENTIAL JOINPOINTS AND THE SCOPE OF THE CONTEXT

From an advise-weaving perspective, joinpoints map to places where weaving takes place – hence the user of "point" in the name. However, from an execution perspective, a joinpoint represents a logical interval of time in a flow of execution. It has a beginning and an end, and advice can be woven into the flow of execution before, after, or around it. This section presents Figure 4 as a pseudo-code for an implementation of *T2* annotations that illustrate five new types of joinpoints for DTPS's: *outer transaction*, *inner transaction*, *resource locked*, *locking*, and *operation*. Each type of joinpoint is in a different color. This section also discusses interesting metadata that advice might want to use, and therefore should be part of joinpoint contexts.

An *Outer Transaction Joinpoint* represents an interval that spans the complete execution of a transaction, starting just before the tentative phase and ending after the completion on the commit phase. This kind of joinpoint would allow a programmer to introduce advice before, after or around an entire transaction. However, because it starts before the beginning of the tentative phase, any "before" advice would not have access to the target transaction's context information. However, it would have access to a parent transaction's context, which would be particularly important for advice before or around sub-transactions.

An *Inner Transaction Joinpoint* is similar to an *Outer Transaction Joinpoint*, except that it starts just after the tentative phase begins and ends just before the commit phase ends. Advice woven before this kind of joinpoint would have access to the target transaction's context.

Resource Locked Joinpoint represents an interval that spans the time when a lock is held, starting after acquiring of the lock and ending just before its release. Advice woven before, after or around this type of joinpoint would have

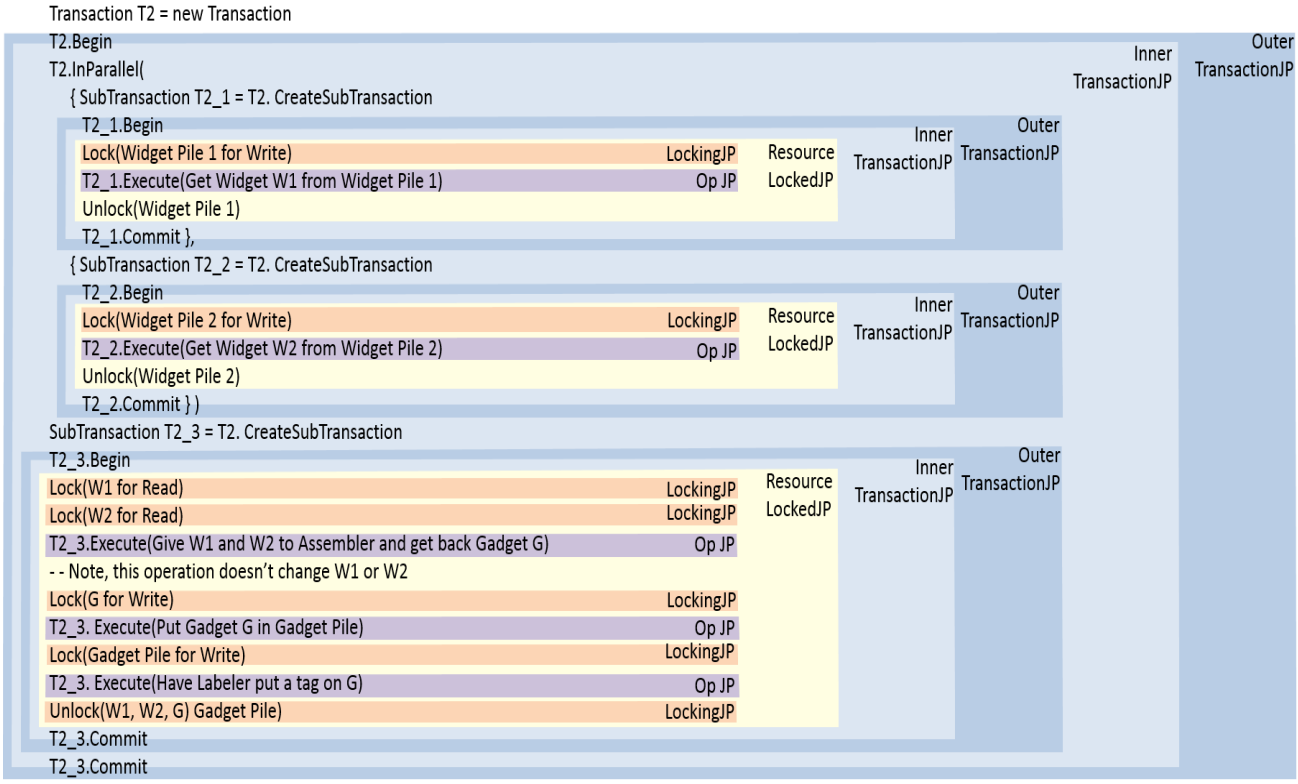


Figure 4 - Pseudo Code for Distributed Version of T2 and the Potential Transaction Joinpoints within the Scope of the T2's Context.

access to metadata about the lock, the associated resources and, of course, the transaction.

Locking Joinpoint represents an interval that spans a lock request. In other words, it begins as a request is made and ends when the request is granted or denied. Advice woven before, after, or around a *Locking Joinpoint* can access metadata about the type of lock being requested or the resource.

Operation Joinpoint is an interval that spans one operation in the execution of the tentative phase of a transaction. Such advice would have to access to metadata about the operation and the affected resources, as well as the transaction as large.

IV. SAMPLE CROSSCUTTING CONCERNS

The number and variety of crosscutting concerns in a DTPS are perhaps infinite. However, for illustrative purposes, we will consider just one here. Imagine that we would like to optimize the Gadget manufacturing system such that Widgets were created just in time, by making sure there are always some Widgets in a pile, but never an excess.

Such flow-control or timing issues could be considered a secondary crosscutting concern to the basic Gadget assembly problem. By talking with the domain experts, we would probably discover a couple of basic rules that govern when the Widget product needs to be speed up or slowed down. An OO programmer could embedded the logic for these rules into the implement of the *Builder*, *Baker*, *Polisher*, or some other set of components. With some skill, it is possible that the OO programmer might even be able to do this in a modular and reusable way.

With transaction aspects, an AOP programmer, however, would have a much similar option. Basically, the programmer would encapsulate the logic for speeding up or slowing down widget production into an aspect, maybe called something like *WidgetProductionSpeedControl*. This aspect would include advice that could be woven before (or around) any operation that accesses a widget pile. The advice's logic would speed up Widget product if the pile was getting too small or slow it down if the pile was getting too large. The aspect would also include a simple pointcut that defined a pattern for all relevant joinpoints. The original application code would not need to be aware of the new production-speed control logic. In fact, because of this obliviousness, it could be tested with or without the speed control functionality without any reprogramming of the system.

V. THE UNIFIED MODEL FOR JOINPOINTS IN DISTRIBUTED TRANSACTIONS

Figure 5 shows part of the UML model, called the *Unified Model for Joinpoints in Distributed Transactions* (UMJDT), which captures the key ideas for the new transaction joinpoints and related context information. The class labeled TransJP is a generalization of the joinpoints discussed in Section III. By definition, each is associated with a StartEvent, but may not have an EndEvent if the interval is still in process. Every TransJP can also reference a context

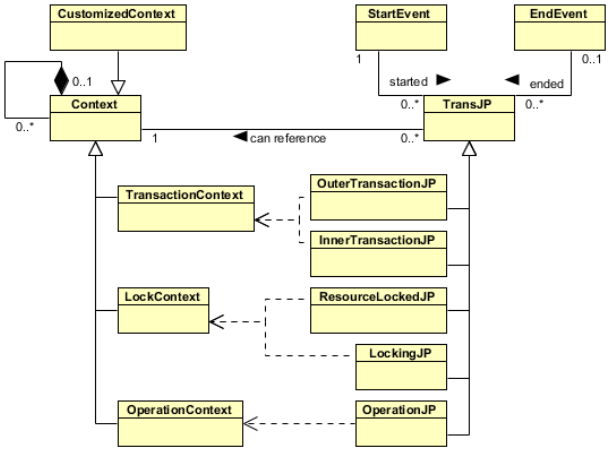


Figure 5 – Part of the Unified Model for Joinpoints in Distributed Transactions

that holds all the relevant statics and runtime information for the joinpoint. Aspect advice will use this context to access a wide variety of information such as operations in progress, resources, and current execution environments.

However, there are three special kinds of contexts, and the actually kind of context that a TransJP directly accesses depends on the TransJP specialization. For example, a LockingJP directly accesses a LockContext.

Contexts can be composited into a hierarchy of objects, as indicated by the recursive aggregation relationship connected to the *Context* class. Although Figure 5 does not show all the possibilities and constraints, a LockContext can be part of a TransactionContext, which could in turn be part of another TransactionContext (i.e., for a parent transaction.)

Contexts may also be extensible or customizable objects. In other words, the base system that makes transaction aspect possible, will provide classes for Context and its three immediate specializations. It also projects hooks for extending those classes, either through specialization, plugs-in, or even other kinds of aspects, so programmers can use context details that are specific to a particular DTPS or DTPS-based applications.

VI. ADVICE WEAVING

Kizcales, et al. introduced the idea of weaving logic for crosscutting concerns into core applications over 15 years ago [2]. Their work stems from even earlier research with inheritance, aggregation, and mix-ins [13]. Like all great ideas, the heart of the weaving solution is relatively straight forward – modularize concerns into first-class constructs, find the right place(s) to introduce appropriate logic from those constructs, and the either insert code that executes the new logic unconditional (because it can be determined to always be needed) or insert code that makes a final decision about executing the new code at runtime.

The challenge for transaction-related aspects is not so much the basic weaving process as it is pulling together all of the relevant data that needs to make up a transaction's

 ISBN: 978-1-61208-367-4

context. Remember, that in a DTPS, the execution of a single transaction is an abstraction that might span many different hosts and be interleaved with the execution of many other concurrent transactions.

So to solve this problem, we propose to build a runtime extension to AspectJ that tracks the start and end events of the TransJP's using low-level distributed aspects. We believe this to be feasible because it is similar to the technique used by CommJ to add communication-related aspects to AspectJ [14].

Although our approach will re-use many of the ideas first prototyped and refined in CommJ, our implementation for the weaving of transaction aspects will have to solve some additional problems not addressed by CommJ. Some of these problems include data-sharing optimizations, like the sharing of context information sharing across hosts only when necessary. Our future work will include research into both static and dynamic analysis techniques for solving these problems.

For the moment, solving the basic weaving and context management problems are sufficiently interesting and potentially beneficial to dominate our immediate attention.

VII. SUMMARY AND FUTURE WORK

This paper presented a foundation for extending AspectJ to support transaction aspects, using joinpoints and context information that is both interesting and relevant to DTPS's. In doing so, it paves the way for the weaving of crossing cutting concerns into high-level program abstractions that span multiple threads of execution and may be interleaved with concurrent execution of similar abstraction.

The main contribution of this paper is simply to identify the set of joinpoints and context information that make the most sense for DTPS's. We have captured this knowledge in a formal model called, *Unified Model for Joinpoints in Distributed Transactions* (UMJDT), as presented its essential parts here.

Our next steps are to a) complete the implementation of the an extension to AspectJ that performs the expected weaving and tracking of context information, and b) perform an preliminary experiment that we hope will provide evidence of improvement in modularization and reuse. To measure the modularity and reuse, we will define an extension to an existing quality model with following new factors: correctness, separation of concerns, understandability, obliviousness, throughput, transaction volume, transaction velocity, and transaction size. Each factor can be measured using metrics, such as diffusion of application, concern diffusion over operations, the number of inter-type declarations, the number of committed transactions, the number of aborted transactions, a rate of data flow during transaction executions, and the length of a transaction design and code, such as the lines of code, the number of operations, the number of components, i.e., classes and aspects, into the transaction, and the weighted operations per component. We also hope to create a toolkit consisting of reusable transaction aspects for common concerns, like performance measuring, logging, exception handling, audit trails, and tracing.

REFERENCES

[1] F. P. Jr. Brooks, "No silver bullet, essence and accidents of software engineering", Computer, vol. 20, no. 4, 1987, pp.10 - 19.

[2] G. Kiczales, et al. "Aspect-Oriented Programming," Proceedings of ECOOP '97, Springer Verlag, 1997, pp. 220–242.

[3] C. Clifton and G. T. Leavens, "Obliviousness, Modular Reasoning, and the Behavior Subtyping Analogy", In Proceedings of the Workshop on Software Engineering Properties of Languages for Aspect Technologies (SPLAT), Workshop at AOSD, December 2003.

[4] J. Gray and A. Reuter, Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Mateo, CA, 1993.

[5] G. Kohad, S. Gupta, T. Gangakhedkar, U. Ahirwar, and A. Kumar, "Concept and techniques of transaction processing of Distributed Database management system," International Journal of Computer Architecture and Mobility. (ISSN 2319-9229) Volume 1-Issue 8, June 2013.

[6] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersen, J. Palm, and W. Griswold, "An Overview of AspectJ". 15th ECOOP 01, June 2001, pp. 327 – 357.

[7] G. Alkhatib and R. S. Labban, "Transaction Management in Distributed Database Systems: the Case of Oracle's Two-Phase Commit," The Journal of Information Systems Education, vol.13:2, 1995, pp. 95-103.

[8] J. Gray, "The Transaction Concept: Virtues and limitations", In Proceedings of the 7th International Conference on VLDB Systems (Cannes, France). ACM, New York, 1981, pp. 144-154.

[9] T. Härder and K. Rothermel, "Concurrency Control Issues in Nested Transactions," Journal of VLDB 2 (1), Jan. 1993, pp. 39-74.

[10] C. Mohan, B. Lindsay, and R. Obermarck, "Transaction Management in the R*· Distributed Database Management System," ACM Trans. on Database Systems, vol. 11, no. 4, December. 1986, pp. 378-396.

[11] G. Colouris, J. Dollimore, and T. Kindberg, "Distributed systems, concepts and design," Addison-Wesley. Fourth edition 2005. ISBN-10: 9780321263544, 2005.

[12] P. A. Bernstein and N. Goodman, "Concurrency Control in Distributed Database Systems," ACM Computing Surveys Vol. 13 No 2, June, 1981, pp. 185-221.

[13] A. Przybyłek, "Systems Evolution and Software Reuse in Object-Oriented Programming and Aspect-Oriented Programming," J. Bishop and A. Vallecillo (Eds.): TOOLS 2011, LNCS 6705, 2011, pp. 163–178.

[14] A. Raza and S. Clyde, "Weaving Crosscutting Concerns into Inter-process Communications (IPC) in AspectJ," ICSEA 2013. Nov. 2013, pp. 234-240. ISBN: 978-1-61208-304-9. Venice, Italy.

[15] A. Hastings, "Distributed Lock Management in a Transaction Processing Environment," In Proceedings of IEEE 9th Symposium on Reliable Distributed Systems, Oct. 1990, pp. 22-31.

[16] B. Gallina, N. Guelfi, and A. Romanovsky, "Coordinated Atomic Actions for dependable distributed systems: the current state in concepts, semantics and verification means," In Proc.

 ISBN: 978-1-61208-367-4

18th IEEE Int. Symposium on Software Reliability (Sweden). Nov. 2007, pp. 29-38.

[17] E. Bodden, "Closure joinpoints: Block Joinpoints without Surprises," In Proceedings of the 10th international conference on AOSD, New York, NY, USA, ACM, 2011, pp. 117-128.

[18] F. F. Rezende and T. Härder, "Concurrency Control in Nested Transactions with Enhanced Lock Modes for KBMSs," In: Proc. 6th DEXA, London, UK, Sept. 1995, pp. 604-613.

[19] H. Kung and J. Robinson, "On optimistic methods for concurrency control," ACM Transactions on Database Systems, Vol. 6, No 2, June 1981, pp. 213-226.

[20] I. Mejía, "Towards a Proper Aspect-oriented Model for Distributed Systems," AOSD '11. ACM New York, NY, USA, March. 2011, pp. 83-84.

[21] J. Eliot and B. Moss, "Nested transactions: and approach to reliable distributed computing Tech," Report MIT/LCS/TR-260, Massachusetts Institute of Technology, 1981.

[22] J. Kienzle, R. Jiménez-Peris, A. Romanovsky, and M. Patiño-Martinez, "Transaction Support for Ada," In Reliable Software Technologies - Ada-Europe, Springer Verlag, 2001, pp. 290 – 304.

[23] K. Donnelly and M. Fluet, "Transactional events," Journal of Functional Programming, v.18 n.5-6, September 2008, pp. 649-706. [doi>10.1017/S0956796808006916]

[24] M. Atif, "Analysis and verification of two-phase commit & three-phase commit protocols," In Proceedings of the 5th ICET, IEEE, Oct. 2009, pp. 326 -331.

[25] N. Dhamir, D. N. Mannai, and A. Elmagarmid, "Design and implementation of a distributed transaction processing system", COMPCON '88, IEEE, New York, Mar. 4, 1988, pp. 185-188.

[26] P. Ram and P. Drew, "Distributed transactions in practice," ACM SIGMOD Record, v.28 n.3, Sept. 1999, pp. 49-55. [doi>10.1145/333607.333613]

[27] R. Banks, P. Furniss, K. Heien, and H. R. Wiehle, "OSI Distributed Transaction Processing Commitment Optimizations," ACM SIGCOMM Comput Commun Rev , Vol. 28, No. 5. ACM, New York, Oct. 1998, pp. 61–75. ISSN: 0146-4833.

[28] R. J. Walker and G. C. Murphy, "Joinpoints as ordered events: towards applying implicit context to aspect-orientation," Workshop on Advanced Separation of Concerns at the 23nd ICSE, 2001.

[29] T. Haerder and A. Reuter, "Principles of transaction-oriented database recovery," ACM Computing Surveys, v.15 n.4, December. 1983, pp. 287-317. [doi>10.1145/289.291].

 ISBN: 978-1-61208-367-4

A Tool Evaluation Framework based on Fitness to Process and Practice

A usability driven approach

Diego Fontdevila

Departamento de Ingeniería
Universidad Nacional de La Matanza
San Justo, Buenos Aires, Argentina
dfontdevila@ing.unlam.edu.ar

Departamento de Ingeniería
Universidad Nacional de Tres de Febrero
Caseros, Buenos Aires, Argentina
dfontdevila@untref.edu.ar

Abstract — **Most current and traditional research on software development tool evaluation focuses on tool capabilities and features following the traditional approach for generic software evaluation. Existing evaluation frameworks and methods address functional and non-functional requirements, constraints, technology, knowledge domain, costs and other acquisition aspects, but such approaches do not account for the context in which work is done. We propose a usability-based framework for tool evaluation in terms of fitness to the development process and practice of their users. Our contribution is a framework for relating ways of working to tool evaluation, and a concrete checklist for performing that evaluation. We present this paper as proof-of-concept of our framework and validate its applicability (but not the evaluation results) by using it to evaluate tools with which we have hands-on experience.**

Keywords-tool evaluation; usability-based framework; process and practice.

I. INTRODUCTION

Most current and traditional research on software development tool evaluation focuses on tool capabilities and features [1][2][3][4][5][6][7][8] following the traditional approach for generic software evaluation [9][10]. Existing evaluation frameworks and methods address functional and non-functional requirements, constraints, technology, knowledge domain, costs and other acquisition aspects, but such approaches do not account for the context in which work is done [11]. For example, tools with the required features might rate well in a feature-based evaluation, but support users poorly by implementing the workflows in a way that does not match the users'. Jadhav and Sonar [12] state that none of the primary studies reviewed address the final step of the selection process: "Purchasing and implementing most appropriate software package". The authors also state "good evaluation practice suggests that some action should be taken to ensure that the selected package performs as well as expected after implementation". The problem with such after-the-fact check of successful evaluation and selection is that mistakes can be very costly; that is why we propose an earlier focus on evaluating the final effectiveness of the implementation beyond traditional tool requirements.

Although research has been conducted on evaluation of technology fitness to context, including software development tools [11], the proposed method is limited to technical issues and maintains a requirements-based approach (the case study is for web services technology). Storey et al. [13] propose collaborative demonstration based tool evaluations, focusing on interoperability and tool integration, not on end user support (the target users are themselves researchers).

We propose a usability-based framework for the evaluation of tools in terms of fitness to the development process and practice of their users. Our contribution lies in providing a framework for relating ways of working to tool evaluation. We present this paper as a proof-of-concept of our framework and validate its applicability by using it to evaluate tools with which we have hands-on experience (this is considered good practice in tool evaluation [6][11]).

The capability of a tool to support the software development process and practices of its users might very well be described in terms of usability, based on the idea that any significant divergence between the tool's model of the work and the actual way the work is performed would make the tool difficult to use. A common scenario for inappropriate process implementation might be having a tool that forces a process so heavy on its users that they abandon it partially or completely. Same with practice, a practice might not fit the process, or a tool might not support the practice appropriately. For example, inconsistencies in code review practice between different teams might turn up in system testing. We consider fitness for use, a key quality notion in any product o process, and extend it to fitness to context, where context is defined in terms of software development process and practice. This work's key contribution is a checklist of specific criteria for evaluating fitness to process and practice, inspired by usability terminology.

First, we present the framework and then apply it to the evaluation of two different tools, one related to Configuration Management practices (Jenkins Continuous Integration Server [23]) and the other to Requirements and Project Management (Pivotal Tracker [24]). These tools have

 ISBN: 978-1-61208-367-4

been chosen by theoretical sampling to provide very different process and practice coverage. Jenkins is a tool that supports a single practice, while Pivotal Tracker covers multiple processes and practices. Our evaluation method assumes the evaluators are familiar with the tool's capabilities and can focus on evaluating their fitness to process and practice. Other methods might be used as a first approach for tool evaluation, to validate basic conformance, followed by applying our approach to the top ranking alternatives.

In Section II, we define several usability principles and propose applications of those principles in the context of software development process and practice. In Section III, we use those relationships to establish tool evaluation criteria based on how well the tool supports process and practice according to those principles. Finally, in Section IV, we apply those principles and criteria to the evaluation of the two tools, and in Section V, we present our conclusions and perspectives.

II. Applying Usability Principles to Process and Practice

At this point, we need to state our working definitions of practice and process (see [14], chapter 4, for a description of the interconnection between process and practice):

Process: It is the flow of work, products and information across the organization that produces value and coordinates the activities of groups with different practices.

Practice: The term practice describes the everyday activities and experience of work. Practices comprise a process, but they can exist without a defined process. If a practice is imposed that is not viable for the people doing the work, that same people will usually redefine the practice.

The reason we choose to focus on practice and process is that process focused perspectives often ignore how the work is actually performed by teams and individuals, and thus loose information that is critical to any improvement effort. In our case, choosing the right tool for the job cannot ignore "the way we do things here to succeed" (to paraphrase the title of [15]).

Usability principles are guidelines for the design of things that are meant to be appropriate for use. They provide guidance for creating usable designs and for evaluating those designs.

Processes and practices are tools that humans use to define, coordinate and execute their activities, and provide a harness for sustainable high quality work (in [16], Alistair Cockburn presents a view of practices as one kind of tool of agile teams). As tools, their success is sensitive to the capacity of people to make use of them. This leads to the following definition:

Process/Practice Usability: A measure of how easy it is to follow a process or practice, including the effort needed to learn, the probability of making mistakes, the cost of such mistakes and the overall satisfaction and motivation promoted by following the practice or process.

In Section III, we present a detailed criteria checklist organized by usability principles to evaluate how well a tool suits the process and practice of its users. The main contribution of this work is the criteria checklist we have created inspired by those usability principles. This checklist is not a usability checklist, for it does not evaluate tool usability, it extends usability terminology to define criteria for fitness to process and practice.

In Section II.A, we offer our own working definition of several usability principles (or heuristics, as they are referred to in [17]), an example of their application to everyday things (the standard view of usability) and a description of how each principle can be applied to processes and practices.

A. Usability Principles

We define usability principles for process and practices. We then apply them to the evaluation of tool fitness to process and practice. In this section, we extend these principles described in [18] (Chapter 1) and [17] (Chapter 5) Heuristics" to define a framework for the software development domain. Here, we define the following principles:

1. Feedback
2. Affordance/natural mapping
3. Matching conceptual models
4. Tolerate mistakes
5. Force function

We have chosen these principles because of the way they resonate with software development process and practice concepts. The initial inspiration for this work came to us with the realization of the importance of the term feedback in the context of both usability and software process improvement. As we explored this idea, we found that other usability principles appeared in both contexts, for example, creating safe work environments by tolerating mistakes is a key agile tenet.

An example of usability heuristic that we have not applied here, because no specific criteria related to it seem applicable to process and practice, is avoid modes [17].

1) Feedback

When we act upon the world, there is a reaction from the world that we can perceive (based on [18], page 27).

In everyday life: When we press a floor button in an elevator, we expect it to light up to confirm that the elevator has been programmed to go to that floor, otherwise we press the button again and again.

In Practice/Process: This principle is key to Shewhart's continuous improvement cycle Plan-Do-Check-Act. The process must be such that it offers continuous feedback so that we can appreciate (and check) the effect of the improvement efforts. Idem for Practice, we need to see the effect of a practice to motivate us to maintain it.

2) Affordance/Natural Mapping

Things should by their outward nature expose what they are for, what their purpose is (based on [18], page 9, in this context affordance means "to be for" something).

In everyday things: A small red iron hammer hung in a red container next to a glass window hardly requires an "Emergency" sign to express that it is there to help us break the window (see [18], page 9, there actually is a psychology

of materials such that glass by itself affords the idea of shattering).

In Practice/Process: Process activities should have obvious effect in the production of quality work by the people involved. In other words, the purpose of all activities should be so clear as to not require explanation beyond the initial adoption phase. As a corollary, process activities should then match exactly the Practice of the people doing the work (i.e., should not make them work in a way they do not believe in).

3) Matching Conceptual Models

Every artifact has an implicit mental model that should match that of the people doing the work (based loosely on [18], page 12).

In Everyday things: People tend to believe that if a coin is bigger, it should be worth more, but that is not always the case.

In Practice/Process: A process should match the view that people participating in it have of their work. A particularly important aspect of this is the coordination of teams with very different practices, like software development and marketing. Each team must have an enabling out-model that allows them to integrate their work (an out-model is our model of something we are mostly ignorant about; in this case, the other team and how they work [15]).

4) Tolerate mistakes

Since mistakes are typical of humans, things should allow us to make mistakes without incurring much rework or frustration.

In everyday things: Pushing one wrong button should not wipe out hours worth of a document we are working in. Systems should recover from mistakes easily and gracefully. When recovery is not possible, or too costly, a force function (the next principle in this Section) might be used to prevent people from making that mistake.

In Practice/Process: Activities should be designed in such a way that we do not have to do them all over again if we make a mistake. Iterative and incremental processes are good examples of this. Practices such as Collective Product Ownership, Collaborative Design, Self-Organized Teams and fluid communication channels around the people working on the product provide excellent means of reducing the impact of mistakes. A culture that fosters exploration and innovation must also "applaud" mistakes as the acceptable cost of trying out new things.

5) Force Function

Things should not allow us to make use of them if there is danger of grave consequences of that use.

In everyday things: Door finger protection for babies are examples of force functions put in place to avoid painful finger injuries.

In Practice/Process: Processes and practices should establish hard boundaries on activities that run the risk of breaking up the team or seriously compromising product quality. For example, the practice of working long hours can drive a developer to burnout, and is typical of processes driven by unrealistic scheduling. The force function might then the opposite practice, disciplined 40hs a week work; it is called Energetic Work and included as one of the core Extreme Programming practices by Kent Beck [19]. Another example is when a person is empowered to break a tie in an argument.

III. A FRAMEWORK FOR TOOL EVALUATION

Software development tools are meant to help to work more efficiently, or to reduce the probability of mistakes, or to record information. The way the tool supports the process and practice of its users (the ones doing the work), its alignment with that process and practice, can determine the appropriateness of the tool and its overall usefulness.

In this Section, we outline a simple framework for tool evaluation based on the usability principles described. First, we describe how tool fitness to process and practice can be evaluated through the usability principles presented. We offer a set of criteria for tool evaluation for each principle, and present an example for each criterion. Finally, we present the concrete steps to be performed for tool evaluation.

A. Tool Evaluation Criteria Checklist by Usability Principle

1) Feedback

A tool should be evaluated according to its capacity to provide feedback on its successful use to support a given practice or process. Possible criteria are:

a) Calculation and display of metrics that reflect the performance of practices or process activities.
b) Validates activity results (e.g., automated test execution, static analysis, and model checkers).
c) Supports collaboration and interaction between individuals that provide the actual feedback. For example, centralized code versioning tools use two styles for coordinating modifications, copy-merge-commit (as in CVS, Concurrent Versioning System, and SVN, Subversion) and lock-modify-commit (as in Microsoft's old Source Safe). The copy-merge-commit style favors parallel modification and fast code integration; thus, providing timely feedback, whereas lock-modify-commit code versioning tools tend to delay integration and thus.

2) Affordance/Natural Mapping

A tool should be evaluated according to how its external appearance suggests its purpose and meaning. Possible criteria are:

a) Uses the user's language to describe practices and process activities.
b) Workflow steps in the tool match the practices and process activities (tools developed in-house tend to work much better in this respect). As an example, Defect Lifecycle Tracking tools need to have a defect lifecycle that matches the one in use by the organization.

 ISBN: 978-1-61208-367-4

c) Is accessible to the people doing the job and they have the appropriate privileges. As a counterexample, only a manager might be allowed to create tasks on which developers must book hours.
d) Supports recording rationale and contextual information to further under-standing between teams, especially in activities that coordinate work between different teams. For example, an Architecture Modeling Tool should record design decision rationale (see a practical application to documentation in [20], "Seven Rules for Sound Documentation", page 24).

3) Matching Conceptual Models

A tool should be evaluated according to the match (or lack thereof) between the tool's model of the work and the actual way the work is performed. Possible criteria are:

a) Supports specific practices and activities that are necessary for the process, practice or methodology: For example, a Scrum planning tool over a general purpose issue tracker, requirements management tool over a document editing requirements plugin, as opposed to end-to-end, generic software engineering tools.
b) Specificity: Tools built for a specific practice tend to match that practice very well and avoid cluttering the interface with low value features (like Jenkins for Continuous integration, described in Example 1 in Section IV.A, or most versioning systems, or UML modeling tools). In practice, such specificity needs to be balanced with good integration with tools that support related practices.
c) Cohesion: The tool supports multiple activities but they are deeply interwoven (e.g., versioning systems and requirements tracking systems, when a developer commits a change to implement a requirement or bug fix, the tool records the relationship between the two, providing traceability).
d) Flexible customization usually allows users to bend the tool to better align it with their own process and practices.

4) Tolerate Mistakes

A tool should be evaluated according to how well it reacts to problems and how helpful it its in guiding or supporting recovery of users towards more effective behavior. Possible criteria are:

a) Does not make judgmental assertions about the meaning of a practice or process activity. For example, in the case of metrics (that provide feedback), a tool should not establish fixed criteria for determining success. In the words of Tobias Mayer "metrics should be used to measure truth — not to measure success or failure. Only measures of truth can be trusted not to incite quick-fix behavior in a team" [21]. This might mean tools driving teams to react to the judgment of the tool by "pushing the dirt under the rug". As a concrete example, a while ago we helped one team to handle a problem in their automated tests. It only took a little time to isolate, but it had driven them weeks ago to disable all tests because they were failing – They had reacted inappropriately to the feedback of their tool and abandoned the good practice of automated testing.
b) Provides means to establish flexible thresholds for status, alarms and notifications, so that teams can configure them according to their context. As an example, tools that generate many e-mails a day with false positive results for a check (e.g., server monitor reporting incorrectly that a server is down) tend to drive teams to ignore any of those e-mails.

5) Force Function

A tool should be evaluated according to the force functions it provides to avoid potentially grave consequences of inappropriate use. Possible criteria are:

a) Supports rules for automatic recognition of inconsistencies. For example, does not allow improper use of a modeling language construct (In the case of UML, a semi-formal language, this can easily become a nuisance).
b) Warns or sets hard restrictions when practices reach unhealthy limits. For example, for a project management tool, a force function might be forbidding team overload.
c) Does not support poor practices because they tend to establish the inappropriate behavior into the team or organization and make it harder to fix in the future. As an example, consider tools that create an economy of compensation (points, money, etc.) for specific activities (e.g., bug fixing). Such practices tend to promote the unthinking pursuit of the compensated activities without regard to the value they provide [22]. Putting a tool in place for that will only make the practice harder to change.

IV. TOOL EVALUATION

In this section, we propose a method for applying the usability principles and criteria to tool evaluation. Evaluation is done for all practices and process activities at the same time to avoid multiple iterations that might make the framework cumbersome.

To evaluate each tool:

1. Identify practices and process activities supported by the tool.
2. For each usability principle
    a. Qualitatively evaluate the tool on each criteria related to the principle.
    b. Rate the tool on Process and Practice support.

The rating provides a simple transformation from the qualitative evaluation of the criteria above into a quantitative rating describing how well the tool follows the principle for the selected practices and process activities. Ratings can be assigned according to the following guidelines:

Low: if the tool fulfills none of the criteria.

 ISBN: 978-1-61208-367-4

Medium: if the tool fulfills one of the criteria.
High: if the tool fulfills two or more of the criteria.

In the following checklists, the evaluation notation is:

√ Complies with criteria (comments for specific practices or activities)

~ Partial compliance or biased interpretation (explanation)

X No compliance or particularly negative design regarding the principle.

A. Example 1: Jenkins Continuous Integration Server

Name: Jenkins
Type: Free Software
Workflow/Phase: Configuration Management
Area of focus: Practice
Main Practices/Activities: Continuous Integration
Description: "Jenkins monitors executions of repeated jobs, such as building a soft-ware project or jobs run by *cron*. Among those things, current Jenkins focuses on the following two jobs:

- Building/testing software projects continuously [...]
- Monitoring executions of externally-run jobs" [23]

Jenkins Evaluation Checklist

1) Feedback

a) Calculates and displays metrics. √ (product build and test status)
b) Automatically validates activity results. √ (automated build and tests)
c) Supports collaboration and interactions that provide feedback. √ (sends e-mails to the whole team when a build fails)

2) Affordance/Natural Mapping

a) Uses the user's language to describe practices and process activities. √ (Main entities are builds, dependencies, jobs).
b) Its workflow steps match the practices and process activities.~ (is centered on one practice, has few process issues).
c) Is accessible to the people doing the job and they have the appropriate privileges. √ (simple authorization scheme, usually developers install and manage it).
d) Provides support to record rationale and other contextual information.~ (allows users to comment almost all entities, but has no focus in rationale).

3) Matching Conceptual Models

a) Supports only practices and activities that are necessary for the process, practice or methodology. √ (Supports only the Continuous Integration practice).
b) Is designed for one specific practice. √ (See previous)
c) Supports cohesively multiple activities when they are deeply interwoven. √ (See previous)
d) Provides flexible customization for better alignment to process and practices. √ (Provides extensive customization and extensions through third-party plugins of which it has a built in market with an many options, besides its own API and being Free Software)

4) Tolerate Mistakes

a) Does not make judgmental assertions about the meaning of a practice or process activity. ~ (a broken build is considered negatively by the tool, but that is defined at the core of the practice, not the tool)
b) Provides means to establish flexible thresholds for status, alarms and notifications. √ (Allows to set custom thresholds on test code coverage, failed build mails can be sent to the author of the change or to the whole team).

5) Force Function

a) Supports rules for automatic recognition of inconsistencies. √ (Checks input values by attempting to use them proactively and offers clear error messages to advice on correcting errors).
b) Warns or sets hard restrictions when practices reach unhealthy limits. X (It does not limit too long builds).
c) Does not promote poor practice. √ (It is a lean tool focused in a single practice without unnecessary or counterproductive features).

TABLE I. JENKINS EVALUATION MATRIX

Process Activity/ Practice	***Feedback***	***Affordance/ Natural Mapping***	***Matching Conceptual Models***	***Tolerate Mistakes***	***Force Function***
Continuous Integration	High	High	High	Medium	High

The results in Table I show overall high scores for Jenkins Continuous Integration server. This fits the fact that it is a tool targeted to a single practice. In other words, if the users follow the practice of Continuous Integration, it is reasonable to expect Jenkins to evaluate as a good candidate for successful implementation.

B. Example 2: Pivotal Tracker Project Management

Name: Pivotal Tracker
Type: Application as a Service
Workflow/Phase: Requirements Management/ Project Management (Scrum)
Area of focus: Process
Main Practices/Activities: Requirements Management/ Project Planning/Project Tracking
Description: "Simple, collaborative project management." [24].

Pivotal Tracker Evaluation Checklist

1) Feedback

a) Calculates and displays metrics. √ (e.g., release burn-down, expected velocity)
b) Automatically validates activity results. √ (if expected velocity does not match actual velocity, it modifies the plan expected end date accordingly; orders requirements by priority automatically)
c) Supports collaboration and interactions that provide feedback. √ (integrates tracking information on completed items from the whole team)

2) Affordance/Natural Mapping

a) Uses the user's language to describe practices and process activities. √ (assuming users are familiar with Scrum)
b) Its workflow steps match the practices and process activities. √ (Planning and Tracking are supported naturally, if a Release plan is in place for the Release Burn-down to work)
c) Is accessible to the people doing the job and they have the appropriate privileges. √ (simple authorization scheme, owner, member, viewer roles).
d) Provides support to record rationale and other contextual information. X (Very little in the way or rationale or contextual information beyond a general description of each user story)

3) Matching Conceptual Models

a) Supports only practices and activities that are necessary for the process, practice or methodology. √ (Supports counting story points for bugs, but strongly discourages it).
b) Is designed for one specific practice. √ (Generally well aligned with Scrum)
c) Supports cohesively multiple activities when they are deeply interwoven. √ (Planning and Tracking)
d) Provides flexible customization for better alignment to process and practices.~ (Very limited, charts in particular)

4) Tolerate Mistakes

a) Does not make judgmental assertions about the meaning of a practice or process activity.~ (Velocity changes in recent iterations affect heavily and automatically the planned outcome of the project, but this is usually good practice after the first few iterations)
b) Provides means to establish flexible thresholds for status, alarms and notifications. X (None)

5) Force Function

a) Supports rules for automatic recognition of inconsistencies. √ (Plans and predicts schedule automatically based on simple velocity metric)
b) Warns or even sets hard restrictions when practices reach unhealthy limits. X
c) Does not promote poor practice. X (Charts and reports are unwieldy)

TABLE II. PIVOTAL TRACKER EVALUATION MATRIX

Process Activity/ Practice	***Feedback***	***Affordance/ Natural Mapping***	***Matching Conceptual Models***	***Tolerate Mistakes***	***Force Function***
Requirements Management	High	Medium	High	Medium	Low
Project Planning	High	High	High	Medium	Low
Project Tracking	HIgh	High	Medium	Medium	Medium

The results in Table II show overall medium-high scores for Pivotal Tracker. This fits the fact that it is a tool targeted to several processes. In other words, fitting multiple user´s processes is more challenging for the tool since it spans a wider range of activities and practices. It still evaluates as a good candidate for successful implementation, but the insights provided by the checklist should be taken into account to reduce risks during tool implementation.

V. CONCLUSION AND FUTURE WORK

The purpose of this paper was to validate the applicability of our usability-based framework for analyzing tool fitness to the user's process and practice (not to validate its results). The principles selected and the criteria proposed to evaluate their concrete application allowed us to conduct the evaluations without obstacle, and the framework did not turn up any inconsistencies during the process. Nonetheless, there is significant overlap between some of them. For example, a tool that has a Matching Conceptual Model will usually have Natural Mapping, and both Feedback and Tolerate Mistakes are related to metrics, although with different perspectives. Overall, the criteria and usability terminology have been effective in supporting the discussion and description of tool fitness to process and practice. One valuable output of the evaluation that complements other evaluation methods based on tool requirements is the qualitative comments produced for each checklist item, which might help implementors to assess areas of risk during the implementation process (e.g., for Pivotal Tracker, item 2.b highlights the need to define release items in the tool if we need to use the release burn-down chart).

Future work includes formal experimentation with tools to validate evaluation results, refinement of principles and criteria, peer feedback and expert validation of the framework, refinement of the evaluation template structure, an in-depth study of the conceptual issues explored in this paper, and the application of the framework to the evaluation of fitness between organizations and practices and processes.

We have learned that the framework is coherent and a viable subject of research, and that the resonance in terminology between usability and process and practice that

 ISBN: 978-1-61208-367-4

inspired this work holds in the practical application to the real tools evaluated.

ACKNOWLEDGMENT

Universidad Nacional de La Matanza and Universidad Nacional de Tres de Febrero fund this research.

REFERENCES

[1] A. Guha Biswas, R. Tandon, and A. Vaish, “A case tool evaluation and selection methodology,” International Journal of Strategic Information Technology and Applications (IJSITA), 4(2), 2013, pp. 48-60, doi:10.4018/jsita.2013040104.

[2] E. Miranda, M. Berón, G. Montejano, M. J. Pereira, and P. Henriques, “NESSy: a new evaluator for software development tools,” In 2nd Symposium on Languages, Applications and Technologies (SLATe'13), Faculdade de Ciências da Universidade do Porto, 2013, pp. 21-38, ISBN 978-3-939897-52-1

[3] E. Anjos and M. Zenha-Rela, “A framework for classifying and comparing software architecture tools for quality evaluation.” In: B. Murgante, O. Gervasi, A. Iglesias, D. Taniar, B.O. Apduhan, Eds. ICCSA 2011, Part V. LNCS, vol. 6786, pp. 270–282, Springer, Heidelberg, 2011.

[4] I. Dalmasso, S.K. Datta, C. Bonnet, and N. Nikaein, "Survey, comparison and evaluation of cross platform mobile application development tools," Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International, 1-5 July 2013, pp. 323-328, doi: 10.1109/IWCMC.2013.6583580.

[5] Center for Assured Software, National Security Agency, US, “CAS Static Analysis Tool Study – Methodology”, December 2012 [Online]. Available from: http://samate.nist.gov/ 2014.05.11.

[6] J. F. Cochran and H.N. Chen, “Fuzzy multi-criteria selection of object-oriented simulation software for production system analysis,” Computers & Operations Research, vol. 32, issue 1, January 2005, pp. 153-168, ISSN 0305-0548, doi:10.1016/S0305-0548(03)00209-0.

[7] X. Franch and J.P. Carvallo, “Using quality models in software package selection,” IEEE Software, January-February 2003, pp. 34–41.

[8] R. Firth, V. Mosley, R. Pethia, L. Roberts Gold, and W. Wood. “A Guide to the Classification and Assessment of Software Engineering Tools” (CMU/SEI-87-TR-010). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1987. [Online]. Available from: http://resources.sei.cmu.edu/ 2014.05.11.

[9] S. Comella-Dorda, J.C. Dean, E. Morris, and P. Oberndorf, “A Process for COTS Software Product Evaluation,” Springer-Verlag, ICCBSS 2002, LNCS 2255, pp. 86–96.

[10] M. Morisio and A. Tsoukias, “IusWare: a methodology for the evaluation and selection of software products,” IEEE Proceedings Software Engineering, 144 (3), 1997, pp. 162-174.

[11] G.A. Lewis and L. Wrage, "A process for context-based technology evaluation: examples for the evaluation of Web services technology," Commercial-off-the-Shelf (COTS)-Based Software Systems, 2006. Fifth International Conference on, Feb. 2006, pp. 13-16, doi: 10.1109/ICCBSS.2006.2.

[12] A.S. Jadhav and R.M. Sonar, “Evaluating and selecting software packages: A review,” Information and Software Technology, vol 51.3, March 2009, pp. 555-563, ISSN 0950-5849, doi:10.1016/j.infsof.2008.09.003.

[13] M.A.D. Storey, S. E. Sim, and K. Wong, “A collaborative demonstration of reverse engineering tools,” ACM SIGAPP Applied Computing Review, vol. 10 issue 1, Spring 2002, pp. 18-25.

[14] J.S. Brown and P. Duguid, The Social Life of Information, Harvard Business School Press, 2000.

[15] I. Gat, “How we do things around here in order to succeed”, Workshop, Agile 2010 Conference, Orlando, August 2010.

[16] A. Cockburn, “What the Agile Toolbox Contains”, Crosstalk Magazine, November 2004.

[17] J. Nielsen, Usability Engineering, Morgan Kauffman Press, 1993.

[18] D. Norman, The Design of Everyday Things, Basic Books, 1988.

[19] K. Beck, Extreme Programming Explained, Embrace Change, Addison-Wesley Professional, 1999.

[20] P. Clements et al, Documenting Software Architecture, Views and Beyond, SEI Series in Software Engineering, Addison-Wesley Professional, 2003 (second edition 2010).

[21] T. Mayer, “Simple Scrum”, Agile Anarchy Blog, [Online]. Available from: http://agileanarchy.wordpress.com/2009/09/20/simple-scrum/, 2014.05.11.

[22] M. Poppendieck, “Team Compensation”, Better Software, July/August 2004, [Online]. Available from: http://www.poppendieck.com/pdfs/Compensation.pdf 2014.05.11.

[23] K. Kawaguchi, et al, “Meet Jenkins”, Jenkins Web Site, [Online]. Available from: https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins, 2014.05.11.

[24] Pivotal Labs, “Pivotal Tracker Features”, Pivotal Tracker Web Site, [Online]. Available from: http://www.pivotaltracker.com, 2014.05.11.

 ISBN: 978-1-61208-367-4

Enhanced Design Pattern Definition Language

Salman Khwaja and Mohammad Alshayeb
Information and Computer Science Department
King Fahd University of Petroleum & Minerals
Dhahran, Saudi Arabia
e-mail: {khwaja & alshayeb} @kfupm.edu.sa

***Abstract*—Design patterns are abstract descriptions of object-oriented designs, which appear repeatedly for a possible high-quality solution. Many design pattern description languages have been proposed. These languages use a combination of a natural language, UML-style diagrams, complex mathematical or logic based formalisms, or eXtensible Markup Language (XML). In this paper, we propose an extension to the Design Pattern Description Language (DPDL), which is based on XML to support composite design patterns. A composite pattern is a special type of design patterns that is formed from a composition of other patterns. Composite patterns capture the synergy arising from the different roles an object plays in the overall composition structure. The enhanced Design Pattern Description Language (eDPDL) is found to be effective in capturing the composite design pattern while representing the whole composite design pattern in a single description.**

Keywords—design pattern language; composite design patterns; UML; XML; DPDL

I. INTRODUCTION

A composite design pattern (also called as composite pattern) is a special type of design pattern that represents a design theme, which keeps recurring in specific contexts. Composite design patterns are the composition of other simple design patterns. The main purpose of the composite design pattern is not to join multiple design patterns but it is to capture synergy in the overall structure of the system. Therefore, composite patterns are more than just the sum of the constituting patterns [1].

One of the purposes of the composite design patterns is to enable a higher level of reuse than individual design patterns and objects [2][3][4]. The modeling of the structure and behavior of the composite design patterns is usually done on object-oriented modeling techniques that use graphical notations such as the Unified Modeling Language (UML) [5][6]. UML has become one of the most widely used general-purpose languages for specifying, constructing, visualizing, and documenting artifacts of software-intensive systems. It provides a collection of notations to capture different aspects of the system and sub-systems under development [7].

The objective of this paper is to propose an extension to the Design Pattern Definition Language (DPDL) [8], which is called extended Design Pattern Definition Language (eDPDL), to be able to express the composite design patterns in a reusable fashion. DPDL was originally created to share design pattern implementation details. DPDL already covers the structural and behavioral aspects of the design pattern and is also flexible. However, DPDL is restricted to specify only the structural and behavioral aspects of a single design pattern. DPDL does not provide any means to specify that a particular component or action is originally part of some simple design pattern. Therefore, the composite design pattern description in DPDL becomes a description of one big complex design pattern instead of the aggregation of few simple design patterns.

This motivated us to propose enhancement to DPDL in order to handle composite design patterns. This will enable us to distinguish the components of individual design patterns and their behavior, which makes the composite design pattern less complex and more understandable.

The paper is organized as follows: Section 2 contains the literature review, in Section 3, the proposed enhancement is presented in Section 4. In Section 5, we present an example to validate the proposed enhancement and finally the conclusion is presented in Section 6.

II. LITERATURE REVIEW

In our literature survey, we could identify only three composite design patterns. These are: Active Bridge, Bureaucracy and Model View Controller (MVC) design patterns.

Active Bridge is usually used in recurring types of frameworks, where the application is needed to be connected with a resource like widget or inter-process communication channel. At the heart of the Active Bridge pattern is Bridge Pattern. Other than that proxy, Observer, Abstract Factory and Factory method design patterns are also used for different components of Active Bridge [9][10].

The second commonly mentioned composite design pattern is Bureaucracy. Bureaucracy design pattern is created using Chain of Responsibility, Composite, Mediator and Observer design patterns. Bureaucracy is also considered as a complex design pattern since it is used in the resource management and interaction of the complex objects. This pattern is highly efficient in developing large application where consistency is important [11]. This design pattern is used in many frameworks including ET++ [10], InterViews [12] and SmallTalk Framework [13].

The most commonly used composite design pattern is Model View Controller (MVC). MVC is also used in designing 3-tier or n-tier architecture frameworks. It is used to handle multiple user interfaces based on the user information or interaction. MVC allows modifying a user interface independent of the application logic or data associated with it [14]. It is usually based on Observer and Strategy design patterns. There are many variations of this design pattern used

 ISBN: 978-1-61208-367-4

in the industry like Model View Presenter (MVP) [15] and Model View ViewModel [16].

Although many researchers have tackled the problem of design pattern description or definition languages but very few worked on the language for the composite design pattern definition or description.

Vlissides proposed a visual notations called Pattern: Role annotation that adds scalability and readability over the Venn Diagram notation [17]. This notation focused on static properties of the design pattern compositions. The notation failed to capture the behavioral aspect of the operations in a design pattern.

Dong et al. [18] used First Order Logic (FOL) theories to specify the structural aspect of patterns and Temporal Logic of Action (TLA) of specify their behavioral aspect. The same techniques were used to specify pattern composition. The specification of the structural aspect of a pattern used predicates for describing classes, state variables, methods and their relations.

Dong et al. also investigated the commutability of pattern instantiation with pattern integration (composition). A pattern instantiation was defined as a mapping from names of various kinds of elements in the pattern to classes, attributes, methods, etc. in the instance. An integration of two patterns was defined as a mapping from the set union of the names of elements in the two patterns into the names of the elements in the resulting pattern. This formal definition of integration is mathematically equivalent to the multiple name mapping approach [18].

Taibi and Ngo [19] also took an approach very similar to the one by Dong et al. Instead of defining mappings for pattern compositions and instantiations, they used substitution to directly rename the variables that represents pattern elements. For instantiation, the variables are renamed to constants, whereas for composition, they are renamed to new variables. The composition of two patterns is then the logical conjunction of the predicates that specify the structural and behavioral properties of the patterns after substitution.

Helm et al. [20] used notion of contracts for describing the behavioral composition of the objects. However, his approach was much broad and not specific to composite design patterns. In addition, it only emphasized on the functional or behavioral aspect of the system and the interactions of the objects in the system.

All of these approaches could be used for composite design patterns but they were not specifically designed for the composite design patterns but were for general composition of design patterns in the system.

Riehle [21] investigated the composite design patterns as a recurring framework. In his technique, he used role-based analysis and described the design patterns composition using role-diagrams. Role-diagrams were supplemented with composition constraints, which specify the set of roles an object may, have to, or must not play.

Dong [22] studied the composite patterns in formal settings. He called composition of two or more patterns as name mapping. He defined name mapping as “classes and objects declared in a pattern with the classes and objects declared in the composition of this pattern and other patterns" [22]. Dong used formal mathematical specification for the structural and behavioral properties of the instance of the composite design pattern.

III. DESIGN PATTERN DESCRIPTION LANGUAGE

DPDL a design pattern description language that provides a flexible and a simple way to express patterns [8]. DPDL covers the maximum possible characteristics of the design pattern in a simple way. Figure 1 shows the high level schema for the DPDL language. At the left most in the diagram is the DesignPattern element; for each design pattern there is a DesignPattern element.

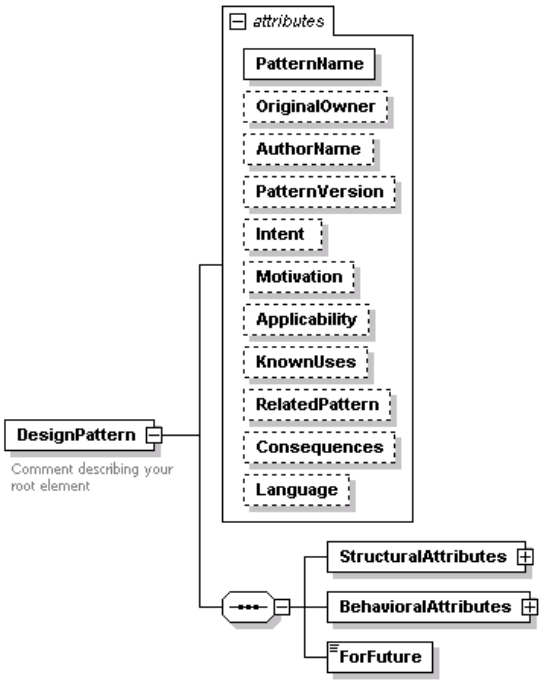


Figure 1. DPDL High Level Schema

As can be seen from Figure 1, the design pattern element has three sub elements; (a) structuralAttributes, (b) behavioralAttributes and, (c) ForFuture. The Structural attribute covers the structural properties of the design pattern. The behavioral attribute defines the behavioral aspect of the design pattern. Finally, ForFuture element is for extending DPDL to add other elements to cover new features of the design pattern in the future.

IV. THE PROPOSED ENHANCEMENTS

Enhancements are made on the original DPDL schema in order to handle the composite design patterns. This section covers these changes. The changes made on the DPDL to handle composite patterns are done on the attributes; no new elements were introduced. Therefore, eDPDL schema is backward compatible; thus, all the existing design pattern instances created using DPDL are still valid on eDPDL.

 ISBN: 978-1-61208-367-4

Two new attributes: isComposite and ConstituentPatterns have been added to DesignPattern element, as shown in Figure 2. *isComposite* attribute is of Yes/No type; if this attribute is Yes that means the description is for a composite pattern thus the designer of the pattern needs to put the design patterns involved in the composite pattern in the *ConstituentPatterns* attribute. *ConstituentPatterns* attribute is of a list type, which means that this attribute can have a list of values delimited by a space.

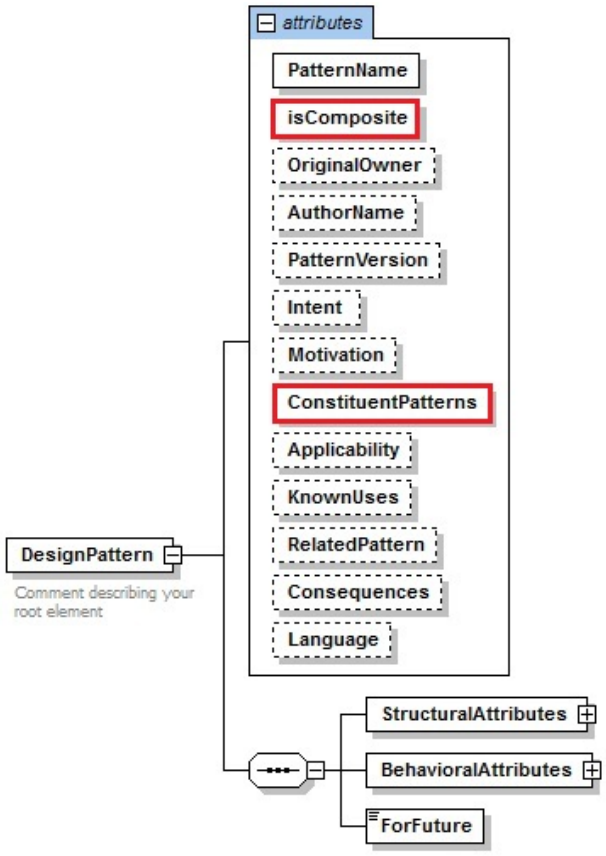


Figure 2. Changes in Attributes of DesignPattern element of DPDL

A. *Changes in StructuralAttribute's Elements*

In the StructuralAttributes element, there are four elements. These elements are Classes, Objects, Operations and Relationships. Each of these elements has a subgroup element. The changes made in StructuralAttributes element are restricted to the changes in the subgroup element of the four main elements of the StructuralAttribute's element. The changes are shown in Figure 3.

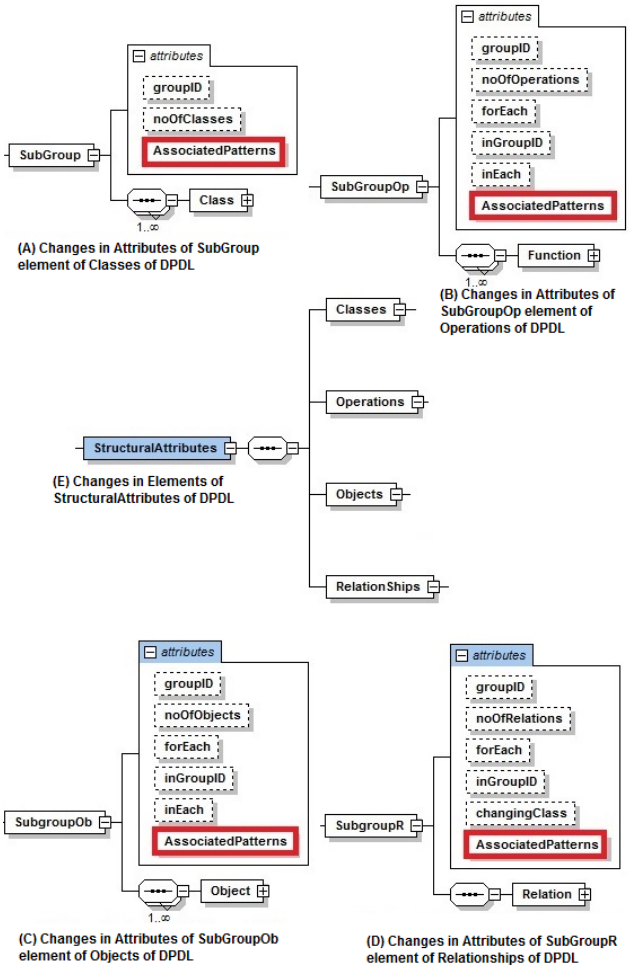


Figure 3. Structural Attributes of DPDL and changes made for eDPDL

As can be seen from Figure 3, AssociatedPatterns attribute (highlighted with a thick rectangle) has been added, hence, each element of class, operation, object or relationship is linked to one or more pattern of the composite design pattern. Therefore, an operation belonging to a particular design pattern in a composite design pattern is mentioned by giving the name of that particular design pattern in the AssociatedPatterns attribute for that particular subgroup element of the operation.

It is also important to mention that attribute name ("AssociatedPatterns") is used in the plural form. This means that multiple design patterns can be listed in this attribute. These patterns can be listed using space delimited. This is done because in some cases a class in a composite pattern might be represent two different patterns in a single composite design pattern.

 ISBN: 978-1-61208-367-4

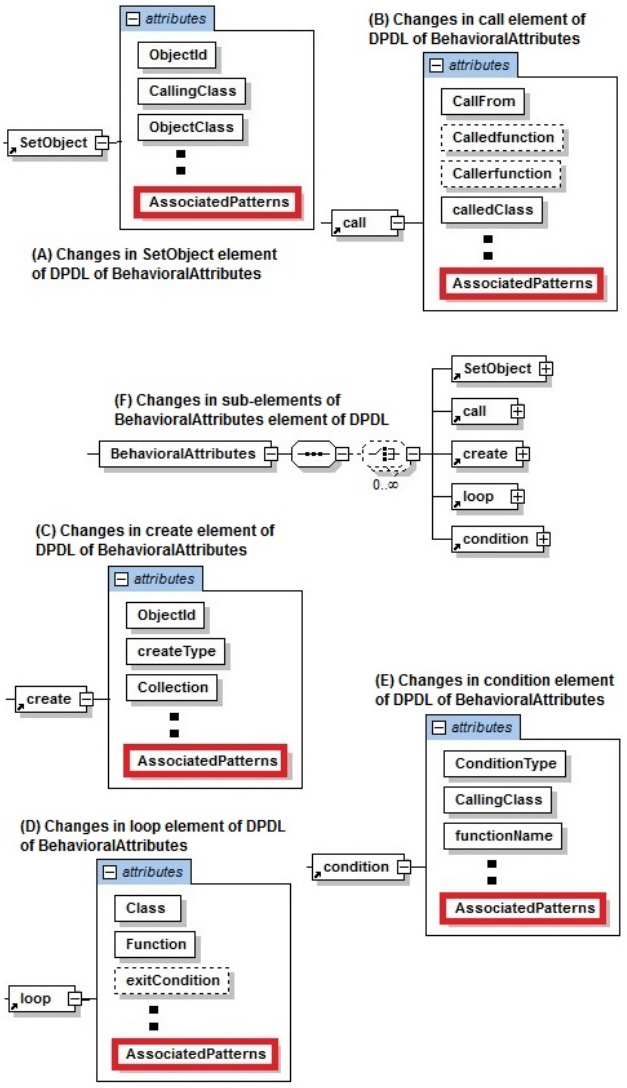


Figure 4. Behavioral Attributes of DPDL and changes made for eDPDL

B. *Changes in BehavioralAttribute's Elements*

The second part of DPDL language is the Behavioral Attributes element. This element has five sub elements describing the behavioral aspect of the design pattern. In all elements related to the behavioral attributes of the design pattern an AssociatedPatterns attribute is added. The changes made for the eDPDL in the DPDL are shown in the Figure 4.

V. EDPDL VALIDATION

Model View Controller (MVC) is a software architecture pattern, which separates the representation of information from the users' interaction with it. There are three types of objects in MVC. Application data is represented by Model, the View is the output or the screen shown to the user, and the Controller handles all the reaction to the input. The Publish-Subscribe protocol is used between model and view - when Model data is changed it will update the View. It also allows attaching multiple Views to the same Model. This is achieved by using the Observer design pattern [23]. Controller implements a particular Strategy for the View, which is similar to the Strategy design pattern. Therefore, this makes MVC a composite design pattern with two design patterns Observer and Strategy. There are different variations of the Model View Controller (MVC) design pattern. Below is one of them [2].

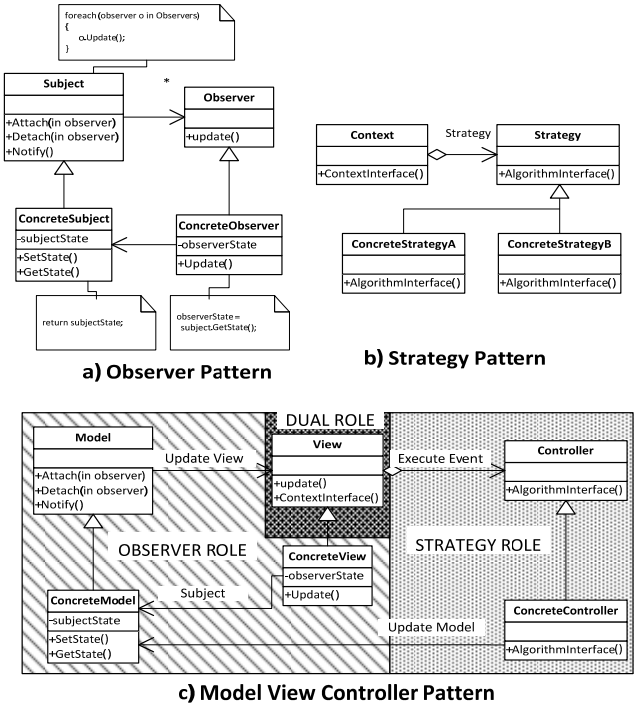


Figure 5. Model View Controller Class Diagram

As can be seen in Figure 5 the shown version of the Model View Controller (MVC) design pattern is composed of Observer and Strategy patterns. The Observer pattern is shown on the left side and the Strategy pattern is on the right side. The view class performs the role of both Strategy design pattern and observer design pattern.

This example shows that there can be a component in the composite design pattern, which acts for more than one design pattern. The update operation in the View class of Model View Controller design pattern is acting in a role of Observer and the contextInterface operation is acting in a role of Strategy design pattern.

We can see in Classes Node that Model Class is defined as part of Observer design pattern and similarly Controller class is defined as link to the Strategy design pattern. However, View Class is shown as part of both Observer and Strategy design group.

Similarly, in Operations group, different operations are also linked with their respective design pattern by listing the pattern in the AssociatedPattern element of the particular operation subgroup. Similarly, same approach is used in the Objects and Relationships Nodes.

 ISBN: 978-1-61208-367-4

```
<StructuralAttributes>
        <Classes>
        <SubGroup groupID="ModelClassGroup" noOfClasses="1" >
        <Class className="Model" isAbstract="Yes" isParent="Yes"
hasConstructor="Yes" classModifier="public" isDerived="No"/>
        </SubGroup>
        <SubGroup groupID="ConcenteModelClassGroup"
noOfClasses="1" >
        <Class className="ConcreteModel" isAbstract="No"
isParent="No" hasConstructor="Yes" classModifier="public"
isDerived="Yes" parentId="Model"/>
        </SubGroup>
        <SubGroup groupID="ViewClassGroup" noOfClasses="1">
        <Class className="View" isAbstract="Yes" isParent="Yes"
hasConstructor="Yes" classModifier="public" isDerived="No" />
        </SubGroup>
        </Classes>
        <Operations>
        <SubGroupOp>
        <Function returnType="Null" containingClassId="Model"
functionName="Attach" functionModifier="public"
inputVariablesType="View" />
        </SubGroupOp>
        </Operations>
        <Objects>
        <SubgroupOb>
        <Object objectName="Views" objectClass="ICollection"
containingClass="Model" objectModifier="private" isList="Yes"
ListType="ICollections"/>
        </Objects>
        <RelationShips>
        <SubgroupR>
        <Relation endClass="View" initiatingClass="ConcreteView"
relationName="Generalization" />
        </SubgroupR>        </RelationShips>
</StructuralAttributes>
```

Figure 6. Example of Structure Attributes of MVC Design Pattern written in DPDL

```
<StructuralAttributes>
        <Classes>
            <SubGroup groupID="ModelClassGroup"
noOfClasses="1" AssociatedPatterns="Observer">
                <Class className="Model" isAbstract="Yes"
isParent="Yes" hasConstructor="Yes" classModifier="public"
isDerived="No"/>
            </SubGroup>
            <SubGroup groupID="ViewClassGroup"
noOfClasses="1" AssociatedPatterns="Observer Strategy">
                <Class className="View" isAbstract="Yes"
isParent="Yes" hasConstructor="Yes" classModifier="public"
isDerived="No" />
            </SubGroup>
            <SubGroup groupID="ControllerClassGroup"
noOfClasses="1" AssociatedPatterns="Strategy">
                <Class className="Controller" isAbstract="Yes"
isParent="Yes" hasConstructor="No" classModifier="public"
isDerived="No"/>
            </SubGroup>
        </Classes>
        <Operations>
            <SubGroupOp AssociatedPatterns="Strategy">
                <Function returnType="Null"
containingClassId="View" functionName="ContextInterface"
functionModifier="public" inputVariablesType="Null" />
            </SubGroupOp>
        </Operations>
        <Objects>
            <SubgroupOb AssociatedPatterns="Observer">
                <Object objectName="Model"
objectClass="ConcreteModel" containingClass="ConcreteModel"
objectModifier="private"/>
            </SubgroupOb>
        </Objects>
        <RelationShips>
            <SubgroupR AssociatedPatterns="Observer">
                <Relation endClass="Model"
initiatingClass="ConcreteModel"
relationName="Generalization"></Relation>
            </SubgroupR>
        </RelationShips>
</StructuralAttributes>
```

Figure 7. Example of Structure Attributes of MVC Design Pattern Written in eDPDL

```
    <BehavioralAttributes>
        <create ObjectId="views" callingClass="Model"
returns="null" Collection="Yes" objectClass="ICollection"
createType="ReadOnly" AssociatedPatterns="Observer"/>
        <call callingClass="Model" returns="null"
CallFrom="function" variableType="{Views}" calledClass="View"
variables="{v}" Callerfunction="Attach" Calledfunction="Add"
AssociatedPatterns="Observer" />
        <call callingClass="Model" returns="null"
CallFrom="function" variableType="{Views}" calledClass="View"
variables="{v}" Callerfunction="Detach" Calledfunction="Remove"
AssociatedPatterns="Observer" />
        <call callingClass="Model" returns="null"
CallFrom="function" variableType="{Views}" calledClass="View"
variables="{v}" Callerfunction="Notify" Calledfunction="Update"
AssociatedPatterns="Observer" />
        <create ObjectId="Controller" callingClass="View"
returns="null" Collection="null" objectClass="Controller"
createType="Readonly" AssociatedPatterns="Observer" />
        <SetObject CallingClass="View" ObjectId="controller"
ObjectClass="Controller" SetTo="Controller"
AssociatedPatterns="Observer" />
        <call callingClass="View" returns="null"
CallFrom="Function" variableType="null"
calledClass="ConcreteController" variables="null"
Calledfunction="AlgorithmInterface" calledThrough="Controller"
Callerfunction="ContextInterface" AssociatedPatterns="Observer" />
        <create ObjectId="ViewState" callingClass="object"
returns="null" Collection="No" objectClass="object"
createType="null" AssociatedPatterns="Observer" />
        <SetObject CallingClass="ConcreteView" ObjectId="Model"
ObjectClass="ConcreteModel" SetTo="Model.ModelState"
AssociatedPatterns="Observer" />
        </BehavioralAttributes>
```

Figure 8. Example of Behavioral Attributes of Model View Controller Design Pattern

Figure 6 shows the structural view of the MVC design pattern written using the original definition of DPDL. Figure 7 and Figure 8 show the structural and the behavioural views of the MVC Design Pattern written in eDPDL respectively. As can be seen, the structural view of the original DPDP does not have AssociatedPatterns tag. Without having this tage, it will be impossible to recognize if the described design pattern is a one large design pattern or the sum of two or more design patterns.

The eDPDL is an extension of DPDL, which not only handles describing the regular single design patterns in a singular fashion but can also describe composite design

 ISBN: 978-1-61208-367-4

pattern as a combination of two or more design patterns. This makes it clear if the design pattern is a composite design pattern or not. The original DPDL description cannot differentiate between composite design patterns and single design pattern. eDPDL is also backward compatible, that is all design pattern which were described based on DPDL schema will work on the schema of eDPDL without any change.

VI. CONCLUSION AND FUTURE WORK

Composite design patterns are usually handled through UML or formal mathematical notations, which are either too complicated or they do not cover the roles and operations comprehensively for the composite design patterns. Thus, the roles that the classes, operations, and attributes play in the pattern get lost. To accomplish the goals of the design pattern, pattern related information becomes important. If this information is not explicitly, the designers are forced to communicate at the class and object level, instead of the pattern level [24].

In this paper, we proposed an extension to DPDL to handle the composite design patterns. The proposed extension, eDPDL, adds attribute to DPDL to handle composite patterns in an easy and efficient way. An example was provided and we found that eDPDL is effective in handling composite design patterns and is also easily understandable as it is built on XML.

Our future research includes extending eDPDL to include other design patterns such as security. We also plan to provide an automated tool to fully support eDPDL.

ACKNOWLEDGEMENT

The authors would like to acknowledge the support provided by the Deanship of Scientific Research at King Fahd University of Petroleum and Minerals, Saudi Arabia.

REFERENCES

[1] D. Riehle and H. Züllighoven, “Understanding and using patterns in software development,” Theor. Pract. Object Syst., vol. 2, 1996, pp. 3-13.

[2] A. Shelest. Model View Controller, Model View Presenter, and Model View ViewModel Design Patterns. Available: http://www.codeproject.com/Articles/42830/Model-View-Controller-Model-View-Presenter-and-Mod. [Retrieved 2014: 18 August 2014].

[3] P. Alencar, D. Cowan, J. Dong, and C. Lucena, “A pattern-based approach to structural design composition,” in Computer Software and Applications Conference, 1999. COMPSAC'99. Proceedings. The Twenty-Third Annual International, 1999, pp. 160-165.

[4] J. Dong, “Design component contracts: model and analysis of pattern-based composition,” Ph. D. Thesis, Computer Science Department, University of Waterloo, 2002.

[5] J. Rumbaugh, I. Jacobson, and G. Booch, "The Unified Modeling Language User Guide," ed: Addison-Wesley, 1999.

[6] J. Rumbaugh, I. Jacobson, and G. Booch, The Unified Modeling Language Reference Manual vol. 2: Addison-Wesley, 2005.

[7] L. Fuentes-Fernahndez and A. Vallecillo-Moreno, “An Introduction to UML Profiles,” in The European Journal for the Informatics Professional, 2004, pp. 5–13.

[8] S. Khwaja and M. Alshayeb, “Towards design pattern definition language,” Software: Practice and Experience, vol. 43, 2013, pp. 747–757.

[9] P. M. Yelland, “Creating host compliance in a portable framework: a study in the reuse of design patterns,” ACM SIGPLAN Notices, vol. 31, 1996, pp. 18-29.

[10] A. Weinand and E. Gamma, “ET++–a portable, homogenous class library and application framework,” Computer Science Research at UBILAB, 1994, pp. 66-92.

[11] D. Riehle, “Bureaucracy,” 1997,

[12] M. A. Linton, J. M. Vlissides, and P. R. Calder, “Composing user interfaces with InterViews,” Computer, vol. 22, 1989, pp. 8-22.

[13] D. Riehle, B. Schäffer, and M. Schnyder, “Design of a Smalltalk Framework for the Tools and Materials Metaphor,” Informatik/Informatique, vol. 3, 1996, pp. 20-22.

[14] F. Buschmann, K. Henney, and D. C. Schmidt, Pattern Oriented Software Architecture: On Patterns and Pattern Languages vol. 6: Wiley, 2007.

[15] M. Potel, “MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java,” Taligent Inc, 1996,

[16] J. Smith, “WPF apps with the model-view-ViewModel design pattern,” MSDN magazine, 2009,

[17] J. Vlissides, "Notation, Notation, Notation. C++ Report," 1998.

[18] J. Dong, T. Peng, and Y. Zhao, “On Instantiation and Integration Commutability of Design Pattern,” The Computer Journal, vol. 54, 2011, pp. 164-184.

[19] T. Taibi and D. C. Ngo, “Formal specification of design pattern combination using BPSL,” Information and Software Technology, vol. 45, March 2003, pp. 157-170.

[20] R. Helm, I. M. Holland, and D. Gangopadhyay, “Contracts: specifying behavioral compositions in object-oriented systems,” in European conference on object-oriented programming on Object-oriented programming systems, languages, and applications, 1990, pp. 169-180

[21] D. Riehle, “Composite design patterns,” 1997, pp. 218-228.

[22] J. Dong, P. S. Alencar, and D. D. Cowan, “Ensuring structure and behavior correctness in design composition,” in Engineering of Computer Based Systems, 2000.(ECBS 2000) Proceedings. Seventh IEEE International Conference and Workshopon the, 2000, pp. 279-287.

[23] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software: Addison Wesley, 1994.

[24] J. Dong, “UML Extensions for Design Pattern Compositions,” Journal of Object Technology, vol. 1, 2002, pp. 151-163.

 ISBN: 978-1-61208-367-4

Model Transformations for the Automatic Suggestion of Architectural Decisions in the Development of Multi-Layer Applications

Jose Garcia-Alonso
Quercus Software Engineering Group
Centro Universitario de Merida
Merida, Spain
Email: jgaralo@unex.es

Javier Berrocal Olmeda
Juan Manuel Murillo
Quercus Software Engineering Group
Escuela Politecnica
Caceres, Spain
Email: {jberolm, juanmamu}@unex.es

***Abstract*—Multi-layer architectures have become one of the most widely used architectures for enterprise application development. Among other reasons, this is due to the proliferation of development frameworks simplifying the implementation of applications based on such architectures. However, the design of these architectures poses a significant challenge to the software architect, mostly due to the large number of design patterns and development frameworks that can be used in the development of these architectures. The present work proposes a set of model transformations to automatically suggest the design patterns and frameworks best suited to satisfy both the functional and non-functional requirements of the system. This technique is part of a broader procedure to facilitate the software architect's task of converting the preliminar design of an application into a specific design tailored to the software architecture.**

Keywords*—*Multi-layer architectures; design patterns; development frameworks; model transformation; architectural decisions.

I. INTRODUCTION

The layer architectural pattern allows software architects to decompose a system into decoupled components called layers. Each layer provides services to the layer above and uses the services of the layer below. The use of this pattern benefits the modifiability, portability, and reusability of the final system [1]. Therefore, multi-layer architectures are those in which the system has been decomposed into two or more decoupled components in a vertical manner.

These architectures are one of the most common solutions to develop enterprise web applications, since they allow developers to focus on the application's business logic instead of its structural details. However, the responsibility for the effective use of these architectures lies with each individual development team [2]. Specifically, the figure of the software architect takes on particular importance since the architecture plays a very important role in the way the application will be developed [3].

Thus, a development success will largely depend on the architect's experience, expertise, and skill in avoiding the introduction of potential errors [4]. Defining the architecture requires the architect to follow an arduous and complex process for getting information on the system requirements and for making decisions about how to structure the application to comply with them [5]. First, the architect has to acquire a great knowledge on the requirements and the relationships between them [6]. Subsequently, the knowledge extracted from the analysis of the requirements is used as the basis for making decisions about how to structure the system [7]. This implies that the architect cannot make these decisions based on a single requirement; she must have a complete view of all the requirements and how they interact. This conjuncture complicates the architect's work and exposes her to situations in which a misinterpretation can lead to the selection of an incorrect architectural pattern.

This situation gets even more complicated due to the close relation between architectural patterns. The application of a given pattern favors the selection of other patterns [8]. Therefore, the incorrect selection of a pattern can lead the architect to make incorrect decisions during the refinement of the architecture. This may cause the final design to fail the requirements of the system, jeopardizing the success of the project. Development frameworks, one of the most used tools in complex software development [9], complicate this problem. The increasing amount of frameworks and their rapid evolution rate [10] make it really difficult to keep up-to-date knowledge about them.

In this paper, a set of model transformations is presented to automatically suggest the architectural decisions best suited for each project. The transformations take as input the initial design of a system, including both functional and non functional requirements, and provides a set of architectural decisions, including design patterns to be applied and development frameworks to be used in the development, that would help the system meet its requirements. This work forms part of a broader proposal that covers the entire process of designing multi-layer applications.

The rest of this communication is organized as follows. Section 2 motivates this work by introducing the process of which the presented transformations are part of. Section 3 details the proposed transformation for automatically suggest architectural decisions. Section 4 specifies the validation performed over the transformation. Section 5 gives a review of the most significant related work. Finally, Section 6 presents the conclusions to be drawn from this work, and some indications of future work planned in this line of research.

II. MULTI-LAYER ENTERPRISE APPLICATIONS

Figure 1 shows a complete diagram of the process proposed for the development of framework-based multi-layer applications.

It shows how the proposed process begins with the preliminar design, normally consisting of a use case diagram and

 ISBN: 978-1-61208-367-4

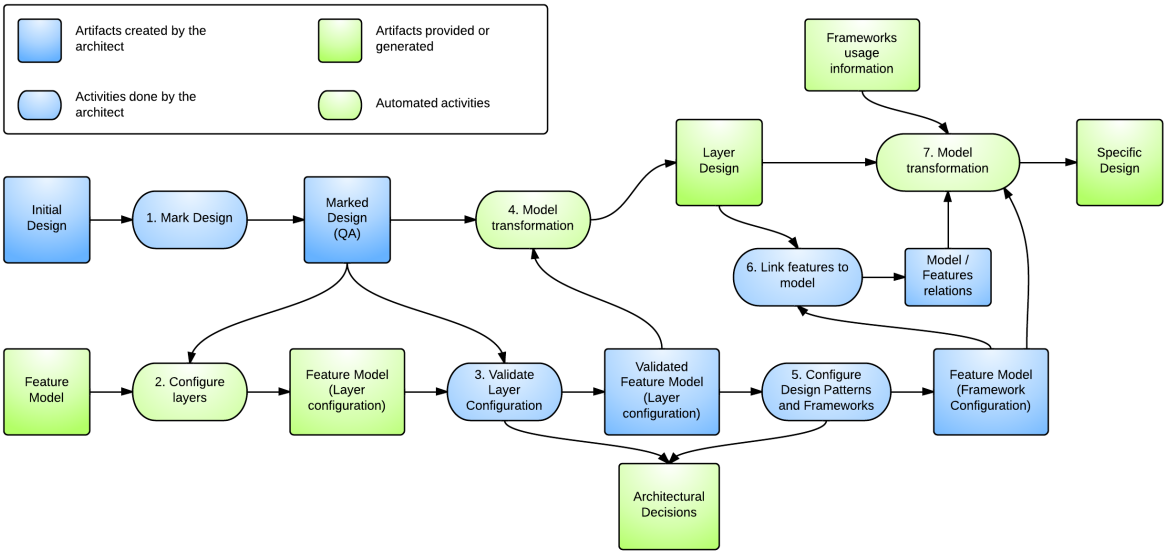


Figure 1. The multi-layer application development process.

multiple activity diagrams representing the behaviour of those use cases. In activity 1, this design has to be refined by the architect or requirements experts to include information about the quality attributes of the system.

Usually, the relationship between functional and non-functional requirements are not explicitly detailed [11]. To make these relationships explicit, the architect or the requirements expert mark the preliminary design with information about the quality attributes to be met by the application. The technique used to accomplish this marking is described in more detail in another paper by Berrocal et al. [12].

Once the architect has the marked design, the next task is to select the layers into which to split the application, activity 2 in the diagram. In order to simplify this task, the process offers to the architect an initial selection of layers. This initial selection is based on the preliminary design and the information added by the marks. However, is the architect who must refine, validate or reject it based on other criteria such as technological limitations, type of project, client, etc. This task is done in the activity 3 in the diagram, more details on the decision-making process followed by architects may be found in [13].

Once the layers have been selected, the initial design can be refined to adapt it to them. This adaptation is performed by a transformation of the model that takes as input the initial design and the configuration of the feature model. This correspond to activity 4.

Feature modeling is one of the most extensively accepted techniques for modeling variability [14]. The specific model used in the present work follows the approach of Cardinality Based Feature Modeling, a widely used technique with proven usefulness in working with development frameworks [15].

To use a feature model as input or output for models transformations it needs to conform to a clearly defined structure or some sort of “metamodel”. This structure must, however, be flexible enough to incorporate both the existing architectural and technological elements and any new ones that may arise in the future. For the model to have these features, we performed a study of some of the most used development frameworks (including Spring, Hiberate, Struts, JSF, CXF, Axis, DWR, etc.). More details on the analysis performed for the creation of the feature model and the decisions made for its creation may be found in [16].

At this point in the process, the architect must specify the design patterns and development frameworks on which to base the final design of the application, activity 5 in the diagram. To make this selection, the architect uses the information contained in the feature model, and then must link each element of the layer design to the architectural decisions that affect it, activity 6 in the diagram.

It should be noted that we propose a specific order for the decision making process, first the layers then the design patterns and finally the development frameworks. However, this order is not fixed and the architect can change it to suit their needs and preferences The abilities exhibited by features model to allow such flexibility were one of the main motivation to choose them as our architectural knowledge representation tool.

Finally, with all the information available, a model transformation is performed to convert the application layer design obtained previously into a specific design for the architectural decisions taken by the architect, activity 7 in the diagram. For this transformation, information is required about the development frameworks to be used. This information is included in the transformation by means of specific models describing the use of a particular technology.

The present work focuses on the model transformations used to offer a set of viable architectural suggestion to the architect; specifically, on activities 2 and 5 in the diagram shown in Figure 1. To accomplish these activities, the transformation use two elements: the feature model containing information about the design patterns and development frameworks that can be used for the development and the preliminary marked

 ISBN: 978-1-61208-367-4

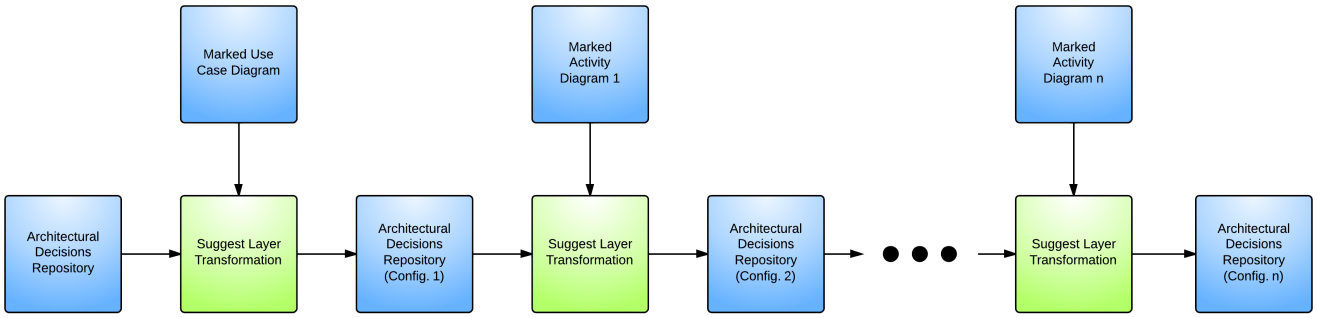


Figure 2. Layer Suggestion Transformation application diagram

design that contains information about the relationship between the requirements and the system's quality attributes.

III. AUTOMATIC SUGGESTION OF ARCHITECTURAL DECISIONS

This section will describe in detail the model transformation used in activities 2 and 5 of the process presented above to automatically provide a set of architectural decisions that can be used for the architect to design a multi-layer architecture that meets the requirements of the system.

A. Automatic layer suggestion

The first model transformation presented is the *Layer Suggestion Transformation*. Figure 2 shows the elements of the process involved in the application of this transformation.

The goal of this transformation is to provide to the architect a possible set of layers to be used in the development of a system. For this, the transformation take as input a feature model containing the set of possible architectural decisions and the initial design of the system marked with information about the QAs it must fulfill. With this information, the transformation generates a copy of the feature model in which the suggested layers has been selected.

As shown in Figure 2, the transformation is designed in such way that it can be applied multiple times, if the initial design of the system is described in several models. Each application of the transformation generates an enriched layer suggestion that can be used as the input of the next application of the transformation. The final result obtained is the set of layers suggested to implement all the elements contained in the different initial design models.

The transformation will suggest a given layer based on specific features found in the initial design of the application or based on the marks containing information about the QAs of the system. Figure 3 shows a fragment of the transformation that suggest the use of a persistence layer if the initial design model contains any DataStore elements.

Figure 5 shows a fragment of the transformation that suggest the use of a security layer if any element of the initial design model is annotated with the given QAs.

This simple set of criteria for layers suggestion can be adapted to meet company policies or architects preferences by

```
1  if(f.isPersistenceConfiguration()){
2      if(UML!DataStoreNode.
           allInstances().size()>0){
3          t.state<-#USER_SELECTED;
4      }
5  }
```

Figure 3. Persistence layer suggestion transformation

```
1  for (stereotype in UCP!Stereotype.
       allInstances()){
2      if(stereotype.name='Authenticity
           ' or stereotype.name='
           Security' or stereotype.name=
           'Confidentiality'){
3          for (useCase in UML!UseCase.
               allInstances()){
4              if(useCase.isAnnotated(
                   stereotype)){
5                  t.state<-#
                       USER_SELECTED;
6              }
7          }
8      }
9  }
10 for (stereotype in AP!Stereotype.
       allInstances()){
11     if(stereotype.name='Authenticity
           ' or stereotype.name='
           Security' or stereotype.name=
           'Confidentiality'){
12         for (activityPartition in
               UML!ActivityPartition.
               allInstances()){
13             if(activityPartition.
                   isAnnotated(
                   stereotype)){
14                 t.state<-#
                       USER_SELECTED;
15             }
16         }
17     }
18 }
```

Figure 5. Security layer suggestion transformation

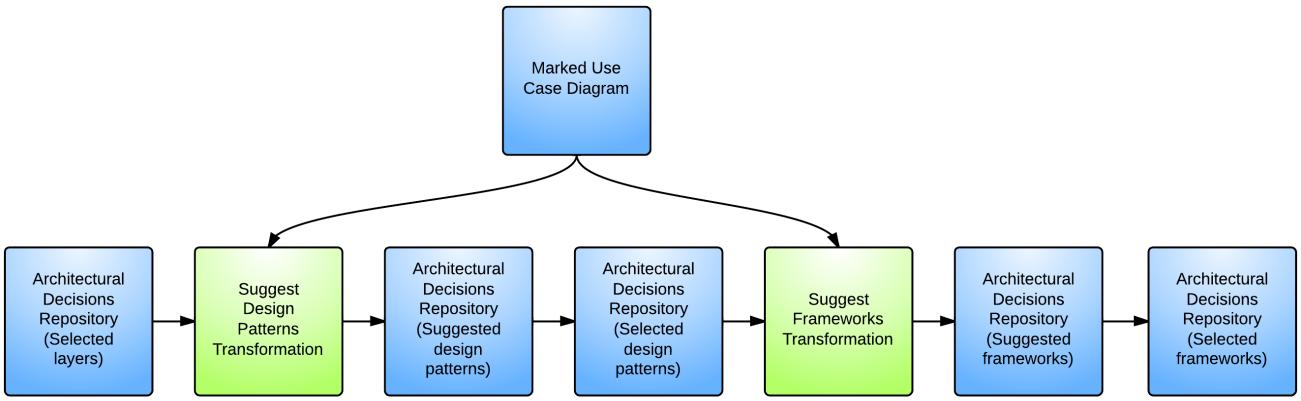


Figure 4. Design Patterns and Frameworks Suggestion Transformation application diagram

```
1  if(thisModule.halfUCAnnotated(
       stereotype)){
2      t.state<-#USER_SELECTED;
3  }
4
5  helper def : halfUCAnnotated(s : UCP
       !Stereotype): Boolean =
6      if (UML!UseCase.allInstances()->
           collect(    ext | ext.
           extensionPoint).flatten()->
           select(exten | not exten.
           getAppliedStereotype(s.
           qualifiedName).oclIsUndefined
           ()).size())/ UML!UseCase.
           allInstances().size() >= 0.5
             then
7            true
8      else
9            false
10     endif;
```

Figure 6. Alternative security layer suggestion transformation

enriching the transformations that suggest each of the layers. Figure 6 shows an alternative to the security layer suggestion that only select such layer if half or more of the use cases of the initial design are marked with the given QAs.

The final product obtained by this transformation is a configuration of the feature model in which the suggested layers are selected. This model will be later used by other transformations to further advance in the development process and can also be used or modified by the software architect.

B. Automatic design patterns and frameworks suggestion

The next model transformation presented is the *Design Patterns and Frameworks Suggestion Transformation*. Figure 4 shows the elements of the process involved in the application of this transformation.

The goal of this transformation is to provide to the architect a possible set of design patterns and frameworks to be used in the development of a system. To do this, the transformation is divided in two. The first one take as input the set of layers selected and the marked use case diagram. With this information, the transformation generates a copy of the feature model in which the suggested design patterns have been selected. The second transformation take as input the previously generated set of selected design patterns and the marked use case diagram and generates a copy of the feature model in which the suggested frameworks have been selected.

The transformation has been divided in two steps in order to give architects the opportunity to refine or validate each level of suggestion independently. Thus, the set of selected design patterns using in the second part of the transformation are not necessarily the ones automatically suggested by the transformation but the ones validated by the architect.

To suggest a particular design pattern or framework the transformation uses the information about the QAs affected by them included in the feature model. This information is checked against the QAs the system must fulfill, as indicated by the marks included in the use case diagram, not forgetting the effect the combination of different design patterns and frameworks has on the final system QAs. Figure 7 shows a fragment of the algorithm used to suggest a framework on the basis of such information.

For each selected design pattern this prioritization algorithm suggest the framework that best helps to fulfill the system QAs based on the framework influence in the QAs and in the relations with the already selected frameworks. The final product obtained by this transformation is a configuration of the feature model in which the suggested design patterns and frameworks are selected. This model will be later used by the last transformation to further advance in the development process and can also be used or modified by the software architect.

The transformation fragment showed in Figure 7 calculates the priority value of a specific framework given the positively and negatively affected QAs by such framework and by its combination with the rest of the frameworks already selected. The framework with the highest priority value is suggested to be used in the development

 ISBN: 978-1-61208-367-4

```
1  --Iteration over the frameworks of a
       selected design pattern
2  for(feature in
       designPatternFrameworks){
3      QAsMeet<-0;
4      --Calculates the priority value
           of the framework based on the
           affected QAs
5      properties<-feature.properties.
           children;
6      for(property in properties){
7          --Positively affected QAs
               added to the priority
               value
8          if(property.name='
               PositivelyAffectedQAs'
               and not property.
               typedValue.oclIsUndefined
               ()){
9              affectedQAs<-property.
                   typedValue.
10             stringValue.split(',');
11             for(affectedQA in
                   affectedQAs){
12                 if(affectedQA.trim()
                       <> ''){
13                     QAsMeet<-
                           QAsMeet +
                           thisModule.
14                     annotatedUC(
                           thisModule.
15                     getStereotype(
                           affectedQA.
                           trim()));
16                 }
17             }
18         }
19         --Negatively affected QAs
               subtracted to the
               priority value
20         ...
21         --Effects of combination
               with previously selected
               frameworks added to the
               priority value
22         if(property.name='
               CombinationAffectedQAs'
               and not property.
               typedValue.oclIsUndefined
               ()){
23             for(relatedFramework in
                   relatedFrameworks){
24                 --Positively
                       affected QAs by
                       relations
                       increase the
                       priority value
25                 --Negatively
                       affected QAs by
                       relations
                       decrease the
                       priority value
26             }
27         }
28     }
```

Figure 7. Framework suggestion transformation

IV. VALIDATION

This section tries to detail the usefulness of the presented transformations. To validate them, they have been applied to two industrial project. Industrial projects were used instead of other validation methods since, to properly ensure their impact and benefits, reasonably large projects were needed.

The two projects involved the development of a medium-size multi-layer application. In each project, the transformations presented here were used and the following features were evaluated: their feasibility, their completeness and the effort required to apply them.

The results obtained evaluating the feasibility of the transformations were very positive. All the information available in the process was used to suggest a set of architectural decision by evaluating every architectural decision posible and choosing the best suited to the system requirements. The only feasibility drawbacks were found on the usability of the transformations. Some of the detected problems were fixed, but additional effort is needed in that regard. In general, the performed validation strongly support that the transformations are feasible, i.e., they can be applied to real-life examples by averagely trained personnel.

The results obtained assessing the completeness of the transformations were encouraging. A significant amount of the architectural decisions taken during the projects were automatically suggested by the transformations. The goal of the process, of which the presented transformations are part of, never was to include the complete range of development frameworks, but to provide a mechanism flexible enough to admit all of them. However, this flexibility has some disadvantages, namely the transformations will never be complete because there will always be a new technology to add. Summarizing, the data collected support that the transformations are complete. They facilitate the use of a broad range of architectural decisions and development frameworks, which is very useful for the development multi-layer applications but they have to be constantly updated to keep up with technological evolution.

The results obtained assessing the effort required to use the transformations are very promising. The use of the transformations causes a small overhead in effort needed, but its effect is diluted in the time saved during development. Additional effort are needed when new architectural decision or technologies have to be included into the transformations. This additional effort can be a major drawback using them. Their potential benefits are shown more clearly where no additional element has to be taken into account. Concluding, the data collected strongly support the effort needed characteristic, indicating that the use of the proposed transformation reduces the total effort spent in the design and development of multi-layer applications.

V. RELATED WORK

In the area of architectural decision making, particularly stand out for their close relationship with our proposal two works of Zimmermann [17][18]. They present a framework for the identification and modeling of recurring architectural decisions, and for converting those decisions into design guidelines for future development projects. In particular, Zimmerman

 ISBN: 978-1-61208-367-4

proposes seven Identification Rules (IRs). These rules have their counterpart in our transformations. The main difference between our work and that of Zimmerman is the use made of those architectural decisions. In his work, the main objective is to gather information for use in future projects. Our focus is on automatically suggesting the best suited decisions to meet the requirements of the system being developed.

In the field of Web application development, Melia et al. [19] propose an extension to the model-driven methods of Web application development. Their proposal is closely related to the present work. Both pursue the same goal – to increase the architect's reliability when using model-driven techniques to design the architecture of a Web application. Nevertheless, their work focuses on RIA development, while ours is intended to encompass the entire class of multi-layer applications. Also, unlike our proposal, the approach in [19] does not allow for control of the technologies used to implement the application, and neither does it provide any mechanism to log and store the decisions made by the architect for later use.

Finally, the studies of Antkiewicz et al. [15] and Heydarnoori et al. [20] are of particular interest in the area of framework-based software development. Antkiewicz's techniques allow the modeling of specific designs for certain frameworks, and these models are then used to generate the source code. Heydarnoori et al. propose a technique for automatically extracting templates for implementing concepts of development frameworks. Unlike our work, the proposed techniques are very code centric, and their creation requires expertise in each framework employed. Our work is aimed at increasing the level of abstraction in the sense of being able to start from a technology-independent design, and progress to obtaining the final specific design by using the decisions made by the architect and model transformations.

VI. Conclusions and Future Work

This paper has addressed the problems facing the software architect when designing a multi-layer architecture. The complexity of these architectures, the complex relationship between functional and non-functional requirements and the high number of development frameworks and how they affect the non-functional requirements complicate the architect's task. We have proposed a technique for simplifying the architectural decision making process by means of the use of a feature model, a marked preliminary design and a set of model transformations to automatically suggest the best suited design patterns and development frameworks. The proposed technique forms part of a broader procedure to address the transition from an initial design of an application to a design adapted to the architecture and technologies selected by the architect. This is a complex process that requires deep technical knowledge of the technologies involved. This complexity can be significantly mitigated by using model-driven development processes.

The next steps related to the architect's decision making and the model transformations presented in this work will be based on improving the prioritization algorithm used to suggest the most appropriate development framework based on the QAs affected by it. This algorithm can be improved by taking into account the frameworks that has not been selected but that can improve the system QAs if they are chosen over the architect manual selection.

Acknowledgments

This work was partially funded by the Spanish Ministry of Science and Innovation under Project TIN2012-34945, as well as by the Autonomous Government of Extremadura and FEDER funds.

References

[1] P. Avgeriou and U. Zdun, "Architectural patterns revisited - a pattern language," in EuroPLoP, 2005, pp. 431–470.

[2] R. S. Pressman, "Manager - What a tangled web we weave," IEEE Software, vol. 17, no. 1, 2000.

[3] L. Northrop, "The importance of software architecture," http://sunset.usc.edu/GSAW/gsaw2003/s13/northrop.pdf, SEI, Carnegie Mellon University, [retrieved: 07, 2014], 2003.

[4] M. Dalgarno, "When good architecture goes bad," Methods & Tools, vol. 17, no. 1, 2009, pp. 27–34.

[5] P. Clements, R. Kazman, M. Klein, D. Devesh, S. Reddy, and P. Verma, "The duties, skills, and knowledge of software architects," in Proceedings of the Sixth Working IEEE/IFIP Conference on Software Architecture, ser. WICSA '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 44–47.

[6] R. Capilla, M. A. Babar, and O. Pastor, "Quality requirements engineering for systems and software architecting: methods, approaches, and tools," Requir. Eng., vol. 17, no. 4, 2012, pp. 255–258.

[7] P. C. Clements, "On the importance of product line scope," in PFE, 2001, pp. 70–78.

[8] N. B. Harrison and P. Avgeriou, "How do architecture patterns and tactics interact? a model and annotation," Journal of Systems and Software, vol. 83, no. 10, 2010, pp. 1735–1758.

[9] I. Vosloo and D. G. Kourie, "Server-centric web frameworks: An overview," ACM Comput. Surv., vol. 40, no. 2, 2008.

[10] M. Raible, "Comparing JVM web frameworks," Jfokus, [retrieved: 07, 2014], 2012. [Online]. Available: http://static.raibledesigns.com/repository/presentations/Comparing_JVM_Web_Frameworks_Jfokus2012.pdf

[11] L. Chung and J. C. S. do Prado Leite, "On non-functional requirements in software engineering," in Conceptual Modeling: Foundations and Applications, 2009, pp. 363–379.

[12] J. Berrocal, J. García-Alonso, and J. M. Murillo, "Facilitating the selection of architectural patterns by means of a marked requirements model," in ECSA, ser. Lecture Notes in Computer Science, M. A. Babar and I. Gorton, Eds., vol. 6285. Springer, 2010, pp. 384–391.

[13] J. Garcia-Alonso, J. B. Olmeda, and J. M. Murillo, "Architectural decisions in the development of multi-layer applications," in Proceedings of the 8th International Conference on Software Engineering Advances, ser. ICSEA '13, 2013, pp. 214–219.

[14] K. Czarnecki, S. Helsen, and U. W. Eisenecker, "Staged configuration through specialization and multilevel configuration of feature models," Software Process: Improvement and Practice, vol. 10, no. 2, 2005.

[15] M. Antkiewicz, K. Czarnecki, and M. Stephan, "Engineering of framework-specific modeling languages," IEEE Trans. Software Eng., vol. 35, no. 6, 2009, pp. 795–824.

[16] J. Garcia-Alonso, J. B. Olmeda, and J. M. Murillo, "Architectural variability management in multi-layer web applications through feature models," in Proceedings of the 4th International Workshop on Feature-Oriented Software Development, ser. FOSD '12. New York, NY, USA: ACM, 2012, pp. 29–36.

[17] O. Zimmermann, "Architectural decisions as reusable design assets," IEEE Software, vol. 28, no. 1, 2011, pp. 64–69.

[18] ——, "Architectural decision identification in architectural patterns," in WICSA/ECSA Companion Volume, 2012, pp. 96–103.

[19] S. Meliá, J. Gómez, S. Pérez, and O. Díaz, "Architectural and technological variability in rich internet applications," IEEE Internet Computing, vol. 14, no. 3, 2010, pp. 24–32.

[20] A. Heydarnoori, K. Czarnecki, W. Binder, and T. T. Bartolomei, "Two studies of framework-usage templates extracted from dynamic traces," IEEE Trans. Software Eng., vol. 38, no. 6, 2012, pp. 1464–1487.

 ISBN: 978-1-61208-367-4

An MDE Approach for Reasoning About UML State Machines Based on Constraint Logic Programming

Beatriz Pérez
Department of Mathematics and Computer Science,
University of La Rioja,
Logroño, Spain.
Email: beatriz.perez@unirioja.es

***Abstract*—Model Driven Engineering promotes models as primary artifacts in the software engineering development process. Such models must conform to a metamodel and held associated constraints which restrict their validity. The verification of models against such requirements becomes therefore a fundamental activity to ensure the quality of a system. In this context, the Unified Modeling Language (UML) constitutes one of the most commonly used modeling languages to represent both static and dynamic aspects of software systems. Nevertheless, while the formalization and analysis of static models has motivated a significant number of proposals, it far exceeds the research done on dynamic models, specially on UML state machines, considered to be the mainstay to represent the dynamics of a system. We have defined a proposal to reason about UML state machines based on Constraint Logic programming (CLP), using *Formula* as model finding and design space exploration tool. We show how to translate a UML state machine model into a CLP program following a Meta–Object Facility (MOF) like framework. Furthermore, we enhance our proposal by giving support for the automatic translation of state machines to Formula specifications, based on a Model Driven Engineering (MDE) approach. The proposed framework can be used to reason and validate UML state machine designs by generating valid sets of execution state configurations and checking correctness properties, using Formula as model exploration tool.**

Keywords*—*UML state machines, OCL, Constraint Logic Programming, reasoning, MDE

I. INTRODUCTION

Model-Driven Engineering (MDE) has been promoted for some time as a solution to handle the increased complexity of software development. In the MDE paradigm, models constitute the cornerstone components during the software development process. Such models must conform to a metamodel and held associated constraints which restrict their validity. Effective verification of models against such requirements becomes therefore a fundamental activity to ensure the quality of a system. In the context of MDE, the Unified Modeling Language (UML) [1] has been widely accepted as the de-facto standard object-oriented software modeling language. In particular, UML is widely used in software design to specify both the static and dynamic aspects of object oriented systems, where UML Class Diagrams and UML State Machines are considered to be the mainstay to represent the statics and dynamics of a system, respectively.

As any other software artifact, software models may contain design flaws. Unfortunately, in some occasions such possible design defects are not detected until the later implementation stages, thus increasing the cost of development [2], [3]. This situation requires a wide adoption of formal methods as well as of verification and validation approaches. In this line, there have been remarkable efforts to formalize UML semantics, in order to address and solve the ambiguity, uncertainty and underspecification issues detected in UML semantics. Nevertheless, while the formalization and analysis of static models has motivated a significant number of proposals [2], [4], [5], [6], [7], [8], [9], [10], it far exceeds the research done on dynamic models, specially on UML state machines or on any other variant of Harel statecharts [11], [12]. In many of such proposals, the formalization and analysis of UML artifacts is accomplished carrying out a translation to another language that preserves the semantics. The resulted translation can be used to reason about the original model by checking predefined correctness properties about the original model [3].

In this paper, we extend the work we presented with I. Porres in [10], [13], which proposes an overall framework to reason about UML Class diagrams annotated with OCL, to give also support to UML State Machines. In particular, in this paper we propose a framework to reason about UML State Machine models based on the *Constraint Logic programming (CLP)* paradigm. As in [10], [13], we use Formula [14] as model finding and design space exploration tool, which is based on algebraic data types and CLP. More specifically, we show how to translate a UML state machine model into a CLP program following a Meta–Object Facility (MOF) like framework. Once a UML state machine model is translated into the Formula language, the Formula tool can be used, for example, to prove the *reachability* of specific states of the state machine or to check for *consistency requirements* of the state machine definition. Furthermore, in order to provide full support for the automatic translation of state machines into Formula, we have included an additional menu option in the Eclipse plugin we presented in [10], to easily and automatically carry out such translation. Our framework can be used to reason and validate UML state machine designs by generating valid sets of execution state configurations and checking correctness properties, using the model exploration tool Formula. We illustrate the usefulness and effectiveness of our approach by applying it to a particular case study.

The paper is structured as follows. Section II provides a brief introduction to UML State Machines and presents a simple case study we use throughout the paper. Section III gives an overview of our proposal for the translation of UML

 ISBN: 978-1-61208-367-4

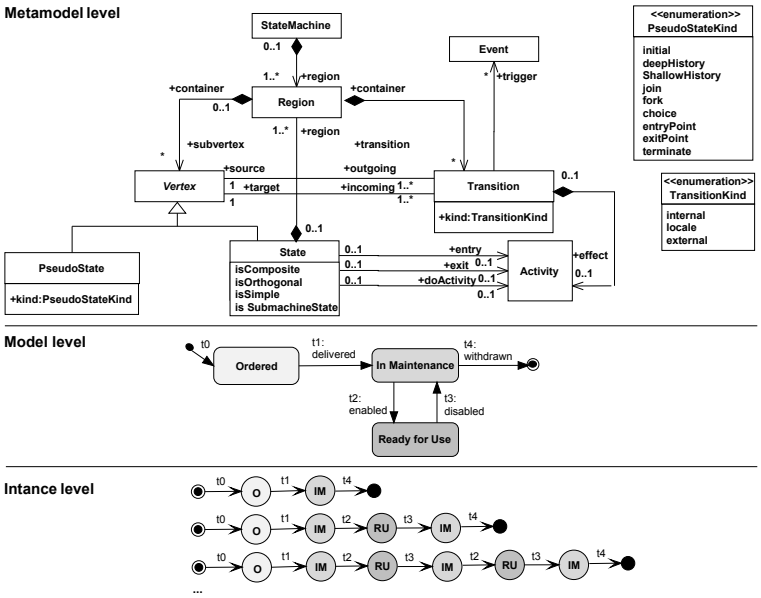


Figure 1: MOF model levels concerning UML State Machines applied to our case study.

state machines to Formula. Section IV explains the application of our proposal, and illustrates the usefulness of our approach by applying it to our case study. Related work is discussed in Section V. Finally, Section VI contains our main conclusions.

II. Backgrounding and Case Study

In this section, we present general background information of UML State Machines, together with the case study we use throughout the paper. In particular, we illustrate UML State Machines with the help of Fig. 1 in which we represent three of the MOF model levels concerning UML State Machines, applied to our case study: the *Metamodel level*, the *Model level*, and the *Instance level*. In particular, we show an excerpt of the UML State Machine metamodel (see the top side of Fig. 1), and the specific state machine model of our case study (see the center side of Fig. 1). This state machine model has been extracted from [15], which we have slightly modified to cover basic aspects of UML state machines for explanation purposes. In particular, this state machine represents the basic states that an object airplane can be in during the course of its life.

As we show in the excerpt of the UML State Machine metamodel depicted on the top side of Fig. 1, a state machine consists essentially of *states*, *transitions* and various other types of vertexes named *pseudostates* [1]. Firstly, *states* denote a situation of objects during which some condition holds. There are three kinds of states: *simple*, *composite* or *submachine*. *Simple* states are characterized by not having substates, while *composite* states are divided into *orthogonal composite states*, to model concurrent behaviors where several states are active simultaneously, and *simple composite states*, to specify that only one of their substates must be active. *Submachine* states are used basically as a way to encapsulate states and transitions so that they can be reused. In our case study, we represent that, over the course of the life of an object plane, it can take up three simple states: *Ordered*, *In Maintenance*, and *Ready for Use*. The valid set of states that the object can be active in, at a specific moment in time during the execution of the state machine, is known as *state configuration*.

On the other hand, a *transition* is the mechanism by which an object leaves a state configuration and changes to a new state configuration. A transition can be triggered by some *event*. In our case study, if the event *deliver* occurs, and the plane is the state *Ordered*, it changes to the state *In Maintenance*, nothing happens if the plane is in any other state than *Ordered*. Particularly, a *transition* is a directed relationship between a source vertex and a target vertex, where these vertexes can be either pseudostates or states. A *pseudostate* is an abstraction used to connect multiple transitions into more complex state transitions paths. There are several kinds of pseudostates (such as initial, join and fork pseudostates). An example of an initial pseudostate is shown in our case study of Fig. 1 depicted by a filled circle, representing the creation of the object plane. Additionally, *composite* states can have one or more *regions* which are considered as simple containers of a connected set of substates, pseudostates and transitions.

Finally, the sequence of state configurations an object can go through during its lifetime is known as *execution trace*. For example, on the bottom side of Fig. 1 we show three of the *execution traces* a plane can be during its lifetime. For a more complete explanation of state machines, we refer to [1].

III. UML State Machines to Formula Translation

Our proposal for reasoning about UML State Machines has some similarities with the approach we presented in [10], [13] for reasoning about UML Class Diagrams, but there is a subtle and essential difference between them. In both proposals we represent the corresponding UML metamodel and model (related to Class Diagrams and State Machines, respectively) in the Formula language, resulting a translation that can be used for several purposes. For example, the resulted Formula specification can be used to rigorously reason about the model's design, by checking predefined correctness properties about the original model such as the lack of redundant constrains. Additionally, the Formula specification can be used to inspect the model, in order to search for conforming object models and to choose those which better fit the domain needs. Nevertheless, while in [10], [13] we aimed at finding sets of classes' instances conforming the class diagram, in the case

 ISBN: 978-1-61208-367-4

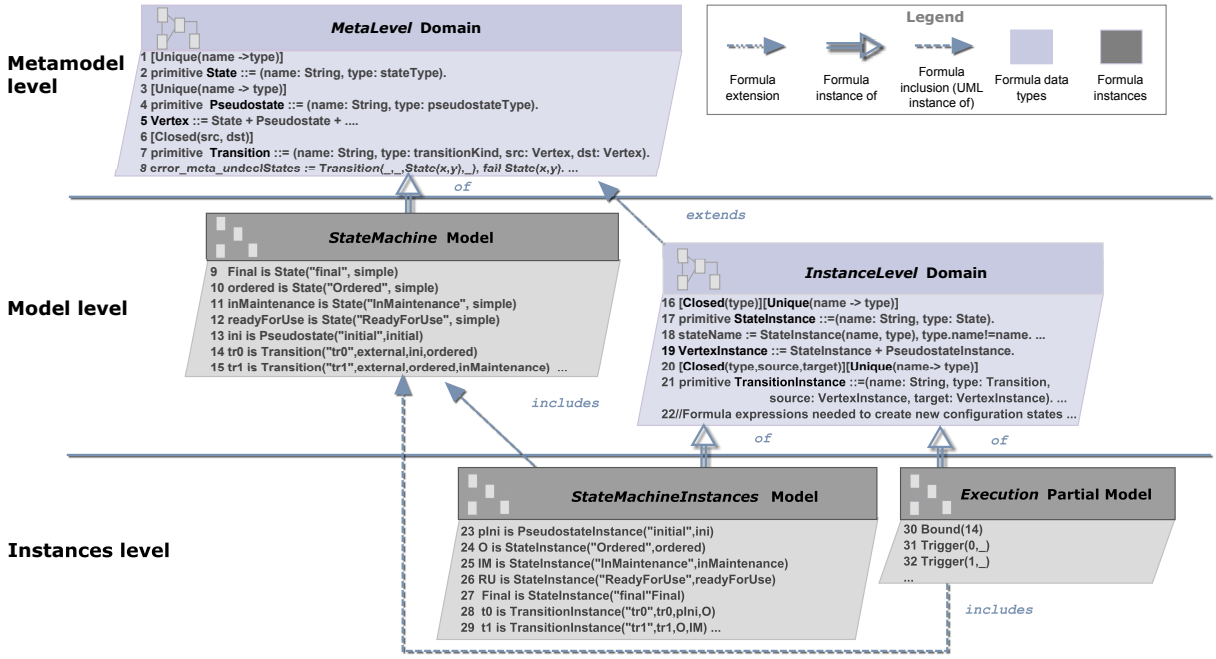


Figure 2: Formula specifications defined for the representation of our case study in Formula.

of UML State Machines we are interested in reasoning and validating UML State Machine designs by generating possible sets of state configurations, simulating valid execution traces.

As we propose in [10], [13], our approach for reasoning about UML State Machines follows a MOF-like metamodeling approach. More specifically, our proposal defines five different Formula units which are distributed along the MOF Metamodel, Model and Instance levels [1]. In order to have a better understanding of our proposal, in Fig. 2 we illustrate the defined Formula units, which are represented by means of rectangles. Furthermore, associated to each Formula unit, we have included, depicted by means of rhomboids, part of the specific Formula expressions that would be defined for representing the state machine of our case study. Next, we briefly explain our proposal for the representation in Formula of a specific UML state machine leaning on this figure.

A. Formula Data Types and Queries

Formula allows to represent a system by using three different units: *domains*, *models* and *partial models*. Firstly, a *problem domain FD* can be specified to formalize an abstraction of the problem that can be used by Formula to reason about the design. This type of units allows to specify abstract data types and a logic program describing properties of the abstraction [14]. For this reason, we have decided to represent the UML State Machine's constructs by means of domains (see *MetaLevel* and *InstanceLevel* domains in Fig. 2).

In particular, a Formula *domain* consists of *abstract data types*, *rules* and *queries*. Firstly, *abstract data types* constitute the key syntactic elements of Formula. They are defined by using the operator *::=*, indicating on the right hand side their properties by means of *fields*. Data types can be labeled in their definition with the `primitive` keyword, defining *primitive constructors*. As an example, in line 7 of Fig. 2 we define the `Transition` data type, which represents the `Transition` element of the UML State Machine metamodel. In particular, it defines several *fields* together with their types (such as the fields `src` and `dst` of type `Vertex`, representing the source and target vertexes of a transition, respectively). If the `primitive` keyword is omitted, the data type definition results in a *derived constructor* (see the definition of the type `Vertex` in line 5, representing the `Vertex` element of the UML metamodel). Data types are used as building blocks for defining Formula expressions (*terms* and *predicates*). Terms are the basis for defining *predicates*, which constitute the basic units of data, used for defining *queries* and *rules*. As an example of the definition of a *term*, in line 8 of Fig. 2 we show the term `Transition(_,_,State(x,y),_)`, representing all instances of the `Transition` term, where the third field is set to a fixed property (`State(x,y)`). The other fields are filled with a do not-care symbol ('_'), so that Formula will find valid assignments. In this way, this term represents any transition whose source state is a specific state (`State(x,y)`).

Based on the defined data types, *rules* and *queries* are specified as logic program expressions, ensuring the remaining constraints [14]. In particular, a *rule* behaves like a universally quantified implication, that is, whenever the relations on the right hand side of a rule hold for some substitution of the variables, then the left hand side holds for that same substitution [12]. The main aim of rules is *production*; they create new entries in the fact-base of Formula, populating previous defined types with facts representing the members in the collection presented in the rule. Rules are specified by means of the operator `:-`, indicating on the left–hand of the expression a simple term and, on the right–hand, the list of *predicates* specifying the rule (an example of a rule is shown later in this section). On the other hand, a *query*, which is constructed by means of the operator `:=`, corresponds to a rule where left–hand side is a nullary construction [12]. A *query* behaves like a propositional variable that is true if and only if the right-hand side of the definition is true for some substitution [12]. In particular, Formula defines in every domain the `conforms` standard query, where all constraints come

 ISBN: 978-1-61208-367-4

together and define how a valid instance of the domain have to look like. Based on the *existential quantification* semantics of queries, we can use them to prove the existence of specific facts in the model. Additionally, the *universal quantification* can be achieved by verifying the negation of a query representing the opposite of the original predicate. For example, to ensure that `Transitions` are not created as connections of undeclared `States`, we firstly need to define a query representing the existence of transitions verifying the opposite (see line 8 of Fig. 2). With this query we are considering such incoherent situation as a valid state. Thus, to verify that such situation is not valid, we need to include the negation ('!') of the query in the `conforms` query of the specific domain.

Taking this into account, the defined domains contain (1) specific data types, which allow us to represent facts and generate reasoning instances of such types, and (2) queries, which restrict the valid system states by specifying forbidden states. Due to space reasons in this paper we mainly focus on explaining the defined data types. In particular, firstly we have created the *MetaLevel* domain, which mainly defines a primitive Formula data type for each meta model element *State*, *Pseudostate*, and *Transition*, together with specific Formula queries representing UML State Machine metamodel constraints (see lines 1 to 7 in Fig. 2). The definition of these types allows the tool to create Formula instances representing specific UML States, Transitions and Pseudostates at the Model level (such as the specific state *Ordered*). We note that, since the representation of the Metamodel level is the same whatever state machine is considered, this Formula domain is defined once and used for each state machine. On the other hand, to be able to represent the information generated during the execution of a state machine (that is, the state configurations which constitute the execution traces, together with the representation of the triggered transitions), we have defined specific types included in a Formula domain *InstanceLevel* (see Fig. 2), which defines types such as `StateInstance` or `TransitionInstance` (see lines 16-17 and 20-21).

B. Formula Data Types' Instances

Having defined the Formula domains with the abstractions of the problem, Formula gives the possibility of creating a *model FM* as a finite set of data type instances built from constructors defined in the associated domain *FD*, and which satisfies all the *FD* constraints [12]. In our particular case, we have defined two different Formula models. Firstly, we have created the *StateMachine* model, which contains the instances of the data types created in the *MetaLevel* domain, and which represent the specific elements of a particular state machine (see Fig. 2). For example, in line 10 of Fig. 2 we show the definition of the element `ordered`, which corresponds to a Formula instance of the constructor `State` defined at the Metamodel level. Secondly, we define the *StateMachine-Instances* model, which contains the instances of the data types defined in the *InstanceLevel* domain. In particular, such instances refer to the state and transition instances that Formula would use as constructors of the execution traces of the specific state machine. For example, in this Formula model we define instances such as *O* (see line 24), which would represent the fact that a specific airplane object has been in the state "Ordered". On the left hand of Fig. 3 we also show graphically the overall instances we would define for the case study. Taking this into account, the *StateMachine* model conforms with the *MetaLevel* domain, while the *StateMachineInstances* model conforms with the *InstanceLevel* domain.

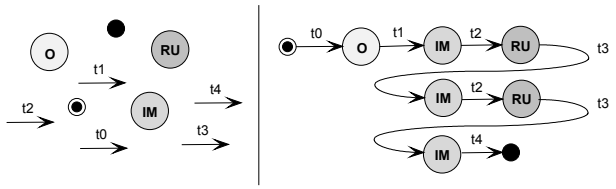


Figure 3: Instance elements and complete execution trace.

C. Logic Instructions to Simulate State Machines' Execution

Up to now, we have established the bases to be able to represent in Formula the UML State Machine metamodel, specific state machines conforming with such metamodel, as well as the concrete instances produced during the execution of a state machine and which would constitute the state machine execution traces. Nevertheless, the defined Formula data types, instances and queries are not enough to allow Formula to reason about the state machine execution, that is, to take such concrete elements and organize them into valid execution state configurations. More specifically, in addition to such instances (see the left hand of Fig. 3) and queries, we provide Formula with specific data types and rules to instruct the tool in the way to reason about such data, so that it is able to generate valid execution traces (such as the one shown on the right hand of Fig. 3). For this reason, we have completed our proposal by defining other two Formula specification blocks.

Firstly, we need to indicate Formula the way in which it has to generate a valid chain of active state configurations' instances which will constitute the valid execution traces. For this task, we have included in the *InstanceLevel* domain the definition of a new data type called `Trigger` (see lines from 3 to 5 in Fig. 4), in order to simulate the triggered of a transition. For this reason, its definition includes a field `t`, referring the moment in which the triggered takes place, and the associated `TransitionInstance` instance (see line 4). We have included the Formula constraint `[Closed(tr)]` to instruct Formula to apply a closed check to instances of the `TransitionInstance` data type, that is, using only the instances of such a type already created in the *StateMachineInstances* model. Based on the `Trigger` type, we define the type `stateConfiguration` to represent a state configuration (see line 7), and which has three fields: (1) `t`, which keeps track in time of the sequence of state configurations, (2) `v`, which refers to the specific active vertex, and (3) `traT`, which refers to the specific transition (`TransitionInstance` instance) which has been triggered to change to that state.

Additionally, in order to construct the chain of state configurations as the transitions are triggered, we have defined a Formula rule (see line 9 in Fig. 4), in order to create new entries of type `stateConfiguration` in the fact-base of Formula. As we have described previously, whenever the relations on the right hand side of a rule hold for some substitution of the variables, then the left hand side holds for that same substitution, and Formula generates the new entry corresponding to the left hand side. Taking this into account, given the current state configuration `stateConfiguration(t,src,traT)` and

 ISBN: 978-1-61208-367-4

```
1 domain InstanceLevel extends MetaLevel {
2  ....
3  [Unique(t ->tr)][Closed(tr)]
4  primitive Trigger ::=(t: Natural, tr: TransitionInstance).
5  triggerNumber:= tri1 is Trigger(t1,tr1), tri2 is Trigger(t2,tr2), t1!=t2,tr1=tr2.
6  primitive Bound::=(t: Natural).
7  stateConfiguration ::= (t: Natural, v: VertexInstance, traT: String).
8  stateConfiguration(0,ini, traT):- ini is PseudostateInstance("initial",
                                     Pseudostate("initial",initial)), traT="__".
9  stateConfiguration (tnext, dst,traTnext) :-
                        stateConfiguration(t, src,traT),
                        Trigger(t, TransitionInstance(tn,_,source,target)),
                        source.name=src.name,
                        target= dst, tnext=t+1,
                        Bound(end), t<end.
10 ...
11}
```

Figure 4: Generation of new state configurations.

the triggered of a transition `Trigger(t, TransitionInstance(tn,_,source,target))` whose source vertex corresponds to the current one (`source.name=src.name`), Formula creates a new fact `stateConfiguration (tnext,dst,traTnext)`, which corresponds to the new state configuration where the new state `dst` is the target vertex of the triggered transition (`target=dst`). The value of the time parameter `tnext` is also incremented by 1 for the following state configuration. Another rule is created (see line 8 in Fig. 4) to get the initial state configuration fact. We also define the `Bound` type to limit the number of transitions triggered during the state machine execution (see `Bound(end), t < end` in line 9).

Secondly, we need to instruct Formula to find valid assignments for the `TransitionInstance` appearances in the `Trigger` elements of the rule in charge of creating new `stateConfiguration` facts (see line 9 in Fig. 4). For these types of tasks, Formula defines another type of Formula units, called *partial models FPM*, in which specify individual concrete instances of the design-space or unknown parts thereof, these latter corresponding to the parts of the model *FM* that must be solved by the Formula tool [14]. For this reason, we have defined a partial model called *Execution* (see Fig. 2), in which we include as many `Trigger` terms as necessary, and which define a do not-care symbol ('_') in the field which corresponds to the `TransitionInstance` instance, so that Formula will find valid transition assignments.

IV. Application and Tool Support

In this section, we briefly explain how to use our framework in practice and apply it to our case study to illustrate its usefulness. Finally, we give some remarks of our plug-in.

The first step to apply our proposal is the translation of the specific state machine we want to reason about into the input specification language of the Formula tool. Such step is carried out by following the guidelines explained in the previous section. Having translated the UML state machine into the Formula language, the Formula finder can be used for different reasoning purposes, such as to prove the *reachability* of states or check the existence of *consistency requirements* in the state machines' definition. In particular, such requirements are represented by means of the definition of new Formula `queries`. Additionally, since the requirements are defined over the execution traces, such queries are included in the `conforms` query of the *InstanceLevel* domain, for their verification. Finally, if the system holds such requirements, the tool returns a state machine execution trace verifying all the established constraints. Otherwise, Formula will have proven that the model is unsatisfiable, that is, not execution trace is possible since some of the constraints become violated. In this latter case, the inconsistencies detected could be taken into account, for example, for the redefinition of the state machine.

In the particular case of using our proposal to prove the *reachability* of states, we can check whether there exists a path which leads to a specific state configuration. A specific use in this line is to find out whether the state machine has a valid execution trace in which the object reaches a final status (that is, there is at least a execution path in which a final status is reached, which corresponds to a *possible existence* property). As an example of application, we can test whether the final state in the state machine of our case study (represented as `stateConfiguration(_,sFinal,_)`) can be reached at some point. In this case, the following query is defined, which is included in the `conforms` query: `q1:= count(stateConfiguration(_,sFinal))=1`. Formula takes as input the state machine specification including this query, and outputs a chain of state configurations proving the reachability of the final state. In particular, Formula returns the following facts, which particularly correspond to the first execution trace depicted in the Instance level of Fig. 1:

```
stateConfiguration(0,pIni,``_'')
stateConfiguration(1,O,``t0'')
stateConfiguration(2,IM,``t1'')
stateConfiguration(3,Final,``t4'')
```

On the other hand, we can also check *consistency requirements* of state machines' definition. More specifically, we refer to consistence from a structural perspective, referring to properties that the model is expected to satisfy irrespective of its semantic content. In particular, we can verify whether the state machine exhibits a number of desired properties, obtaining at the same time the corresponding execution traces proving that the state machine holds such properties. For example, we can check whether an air plane can be available during its life time a specific number of times, obtaining the corresponding trace of state configurations. In particular, this property is checked by defining the query: `q2:= count(stateConfiguration(_,sRFU,_))=`*`number`*.

As for as tooling support, we have taken our *CD2Formula* plug-in presented in [10] to automatically translate specific class diagrams into Formula and we have modified it giving support for UML state machines. Finally, we have included both functionalities in an only plug-in called *UML2Formula*. Again, we have used MOFScript tool [16] which provides support for customizable model–to–text transformations. We use the UML 2.0 metamodel and the specific state machine as the model which can be defined using any UML 2.0 compliant tool that can create models in the XMI format supported by EMF (for example, the UML2 Eclipse plug-in [17] or a UML2 compliant graphical tool). As far as the Formula units generation is concerned, we have defined an only set of MOFScript transformation scripts that generates the different Formula units as stated in our proposal. The defined MOFScript transformations have been integrated into the plug–in, allowing the automatic generation of the Formula specification by means of a menu option the plug–in provides. Applying this menu option to a specific state machine, the plug–in returns a `.4ml` extension file. Later, the specific query

 ISBN: 978-1-61208-367-4

properties to be checked have to be manually included in this file, which Formula will use for reasoning about the model.

V. Discussion and Related Work

In the past decade, there are several works which have used Constraint Logic Programming to formalize UML semantics, being limited those which tackle UML State Machines or on any other variant of Harel statecharts [11], [12], [18], [19]. In particular, there have been some proposals which aim at formalizing UML State machines which have followed a MOF–like approach to a greater or lesser extent. More specifically, authors in [11] focus on Hierarchical Finite State Machines (HFSM), which are a simplified version of UML State Machines, which consider more structural elements (such as concurrent states and pseudostates). The difference between both proposals, besides the different types of modeling languages, lies in the main goal. In particular, authors in [11] give an approach to complete partially specified dynamic models. More specifically, starting from a partial model constituted by unlinked states and transitions, they are able to find a complete state model defined from that partial model and which conforms with the HFSM metamodel. In contrast, our proposal aims at reasoning about specific state machines, not arbitrary ones, that is the reason because it starts from a complete specific state machine model instead of a partial one. In [12], authors present a metamodeling framework based on Formula and reason about *typed graphs*. In particular, they give a metamodel-based approach for representing only the `MetaNode` and the `MetaEdge` elements, at the Metamodel level, and graph `nodes` and `edges`, at the Model level, and finally reason about models. In particular, they apply their proposal to the particular case of state diagrams (where states are nodes and transitions are edges) in order to construct, similarly to the proposal in [11], well–formed state diagrams. In [19] where the author uses Alloy, a textual modelling language based on first-order relational logic, used in other works for analyzing UML class diagrams [18], gives a proposal to simulate states by specifying the notion of state on the model level, in an Alloy model, while the transition between states is given by the invocation to a UML operation.

VI. Conclusion and Future Work

In this paper, we present a framework to reason about UML state machine models based on the CLP paradigm. The main contribution of our work is the translation of a UML state machine into a Constraint Satisfaction Problem following a MOF–like framework. We enhance our approach by providing an MDE-based implementation of our translation proposal, based on our *UML2Formula* plug–in. Particularly, starting from a UML state machine representing the dynamic structure of a software system, our plug–in carries out the automatic generation of the Formula specification corresponding to such UML model, by simply choosing a menu option the plug–in provides. The proposed framework can be used to reason and validate UML state machine designs by generating valid sets of execution state configurations and checking correctness properties, using the model exploration tool Formula.

Our proposal considers basic UML State Machine elements, but the support for other commonly used elements (such as guards or composite states) constitutes a remaining work.

Acknowledgments

This work has been partially supported by the University of La Rioja (project PROFAI13/13).

References

[1] OMG, "UML 2.4.1 Superstructure Specification," document formal/2011-08-06, August, 2012. Available at: http://www.omg.org/. Last visited on August 2014.

[2] A. Calì, D. Calvanese, G. D. Giacomo, and M. Lenzerini, "A Formal Framework for Reasoning on UML Class Diagrams," in *Proc. of the 13th International Symposium on Foundations of Intelligent Systems (ISMIS'02)*, ser. LNCS, vol. 2366. Springer, June 2002, pp. 503–513, ISBN:3-540-43785-1.

[3] J. Cabot, R. Clarisó, and D. Riera, "Verification of UML/OCL Class Diagrams using Constraint Programming," in *Proc. of the 2008 IEEE International Conference on Software Testing Verification and Validation Workshop (ICSTW'08)*. IEEE Computer Society, April 2008, pp. 73–80, ISBN: 978-0-7695-3388-9.

[4] A. Queralt, A. Artale, D. Calvanese, and E. Teniente, "OCL-Lite: Finite reasoning on UML/OCL conceptual schemas," *Data Knowl. Eng.*, vol. 73, pp. 1–22, 2012.

[5] M. Balaban, A. Maraee, and A. Sturm, "Reasoning with UML Class Diagrams: Relevance, Problems, and Solutions – a Survey," March 2007, available online at: http://www.cs.bgu.ac.il/ mira/CDReasoning-07.pdf. Last visited on August 2014.

[6] M. Broy, M. V. Cengarle, H. Grönniger, and B. Rumpe, "Considerations and Rationale for a UML System Model," in *UML 2 Semantics and Applications*, K. Lano, Ed., 2009, pp. 43–60.

[7] J. Osis and U. Donins, "Formalization of the UML Class Diagrams," in *Evaluation of Novel Approaches to Software Engineering*, ser. Communications in Computer and Information Science. Springer, 2010, vol. 69, pp. 180–192, ISBN: 978-3-642-14818-7.

[8] D. Berardi, A. Cali, D. Calvanese, and G. Di Giacomo, "Reasoning on uml class diagrams," *Artificial Intelligence*, vol. 168, pp. 70–118, 2005.

[9] M. Gogolla, J. Bohling, and M. Richters, "Validating uml and ocl models in use by automatic snapshot generation," *Software and System Modeling*, vol. 4, no. 4, pp. 386–398, 2005.

[10] B. Pérez and I. Porres, "An Overall Framework for Reasoning About UML/OCL Models Based on Constraint Logic Programming and MDA," *Intern.Journal on Advances in SW*, vol. 7, no. 1&2, pp. 370–380, 2014.

[11] S. Sen, B. Baudry, and D. Precup, "Partial Model Completion in Model Driven Engineering using Constraint Logic Programming," in *Proc. of the International Conference on Applications of Declarative Programming and Knowledge Management (INAP'07)*, 2007, pp. –.

[12] E. K. Jackson, T. Levendovszky, and D. Balasubramanian, "Automatically reasoning about metamodeling," *Software & Systems Modeling*, pp. 1–15, february, 2013.

[13] B. Pérez and I. Porres, "Reasoning About UML/OCL Models Using Constraint Logic Programming and MDA," in *Proc. of the Eighth International Conference on Software Engineering Advances (ICSEA'13)*, October 2013, pp. 228–233, ISBN: 978-1-61208-304-9.

[14] FORMULA - Modeling Foundations, Website: http://research.micro–soft.com/en-us/projects/formula/. Last visited on August 2014.

[15] H. Baumann, P. Grassle, and P. Baumann, *UML 2. 0 in Action: A Project-based Tutorial*. Packt Publishing, 2005.

[16] MOFScript Home page, Website: http://www.eclipse.org/gmt/mofscript/. Last visited on August 2014.

[17] The Eclipse UML2 project, website: http://www.eclipse.org/modeling /mdt/?project=uml2. Last visited on August 2014.

[18] K. Anastasakis, B. Bordbar, G. Georg, and I. Ray, "UML2Alloy: A Challenging Model Transformation," in *Proc. of the 10th International Conference on Model Driven Engineering Languages and Systems (MoDELS'07)*, ser. LNCS, vol. 4735. Springer, 2007, pp. 436–450.

[19] D. Jackson, "Automating first-order relational logic," *SIGSOFT Softw. Eng. Notes*, vol. 25, no. 6, pp. 130–139, 2000.

 ISBN: 978-1-61208-367-4

Several Issues on the Layout of the UML Sequence and Class Diagram

Oksana Nikiforova, Dace Ahilcenoka, Dainis Ungurs, Konstantins Gusarovs, Ludmila Kozacenko
Faculty of Computer Science and Information Technology
Riga Technical University
Riga, Latvia
{oksana.nikiforova, dace.ahilcenoka, dainis.ungurs, konstantins.gusarovs, ludmila.kozacenko}@rtu.lv

***Abstract* — Models are widely used and are one of the advanced tools of software engineering. Therefore, it is very important that the models and diagrams are well built not only considering their content, but also how they visually represent information, how they are layout. Layout is an important factor considering readability and comprehensibility of a diagram. Providing manual diagram layout is time consuming; it can also be ineffective; therefore, this paper is a research about diagram automatic layout. UML provides a variety of diagrams, which covers all of the system development life cycle steps. The most important UML diagrams are class and sequence diagrams, because they are the main diagrams to present system structure and behavior. We analyze existing layout techniques and algorithms, offer new ones and evaluate them regarding their applicability to class and sequence diagram layout in different modeling tools, how they comply with layout criteria.**

Keywords – UML class diagram; UML sequence diagram; layout algorithm; BrainTool.

I. INTRODUCTION

One of the tasks of software development is to present different aspects of the system before developing the software solution for that system. To solve this task, system modeling became one of the important activities during software development. Models are useful for understanding problems, communicating with everyone involved within the project (customers, domain experts, analysts, designers, etc.), modeling enterprises, preparing documentation and designing programs and databases. Modeling promotes better understanding of requirements, more clear designs and more maintainable systems. Graphical models help to provide a common base for system developers at different levels of system domain and are used at different stages of system abstraction. It is specially pointed to such standardized modeling mean as Unified Modeling Language (UML) [1].

The graphical aspect of modeling language turns developers to an intuitive language semantics and perceptible location of model elements on the diagram. Thus, modelers have to decide two main tasks during creation of the diagram: to think of how to present system functionality by diagram elements and to invent an optimal placement of diagram boxes and wires. Thus, the systematic approach to elements placement within the diagram, which is specified as a task of diagram layout, plays an important role in completing the task of system modeling. This paper tries to solve the problem of diagram layout in correspondence with the most used UML diagrams, namely the UML class diagram and the UML sequence diagram. The goal of the research is to offer layout algorithms for both diagrams, to implement the presenting algorithms within the BrainTool [2] modelling tool, which gives an ability to generate UML diagrams from the two-hemisphere model [3].

The paper is structured as follows. The next sections introduce requirement set to layout the UML sequence and class diagrams. The algorithm based on the defined requirement sets is described in the second and the third section. The fourth section gives a brief overview of the related work and compares our solution with the existing ones. We discuss about the present research and state the direction for the future in the conclusion of the paper.

II. LAYOUT ALGORITHM FOR THE UML SEQUENCE DIAGRAM

Basically, the software system development starts with the business information gathering and presenting it in the form suitable for further software system modeling. Then, this presentation of business information has to be transformed into the model, which in object-oriented manner for software development requires to present objects to interact in the form of UML sequence diagram [1]. It shows objects, their lifelines and messages to be sent by objects-senders and performed by object-receivers and is used to present dynamic aspect of the system. The dynamic of interactions is defined by an ordering of the messages. It serves as a basis for definition of operations performed by objects to be grouped into classes, as well as to present and to verify a dynamic aspect of class state transition. UML sequence diagram is a popular notation to specify scenarios of the processing of operations as its clear graphical layout gives an immediate intuitive understanding of the system behavior. UML sequence diagram is stated as one of the ambiguous UML diagrams, with an implicit and informal semantics that designers can give to basic sequence diagram as a result of this conflict.

The time aspect plays the most important role and helps to organize messages in correct sequences. Vertical axis is used to display time, the beginning of the diagram is at the top and it is read downwards. Sequence diagram can consist of many different elements; however, the authors will use only those, which can be acquired from the two-hemisphere model, which is a kind of initial presentation of the problem domain in the model form, which consist of process model interrelated with conceptual model. It is possible to generate UML sequence and class diagram from the two-hemisphere model based on the direct transformation of diagrams, which

 ISBN: 978-1-61208-367-4

are already explained in [3][4]. To enable implementation of the model transformation by a tool, it is necessary to have an algorithm for element placement after the transformation execution.

A. Layout Requirements for the UML Sequence Diagram

Table 1 shows the list of criteria for layout the elements of the UML sequence diagram in descending order of their importance. Criteria are marked with SD identifier. General layout criteria result from the theory of perception [5]. Specific diagram like the UML sequence diagram has additional criteria, e.g., "slidability". There are six perceptual principles referring to organization of diagram elements, when the elements are considered as a group [6]. These principles are acquired from Gestalt theory [5]. There are three more principles related to perceptual element segregation. All of these Gestalt theory principles are considered as aesthetic criteria. General aesthetic criteria are widely discussed in [7][8][9].

TABLE I. CRITERIA FOR LAYOUT OF THE UML SEQUENCE DIAGRAM

ID	Name of criterion	Description
SD0	Precise sequence of messages	Notational convention of the UML requires to display messages in the order they are being sent.
SD1	Avoid object and lifeline overlapping	When objects or lifelines are overlapping it is hard or sometimes impossible to read the diagram.
SD2	Elements to be arranged orthogonally	Sequence diagram is an example of orthogonal diagram - message arrows are situated horizontally (typically) and lifelines - vertically.
SD3	Diagram flow	It is very important to layout elements by creating obvious flow - visible start and end of the diagram, easier to follow the elements and read the diagram. The first message is located at the top left corner of sequence diagram.
SD4	Minimize crossings	In the sequence diagram message arrows should not cross at all, therefore with crossings is understood message arrow crossings over lifelines and number of this kind of crossings should be reduced.
SD5	Message arrow length minimization	To make the diagram more comprehensible and the area smaller, the message arrow length should be minimized
SD6	Reduction of long message arrow number	It is difficult to follow long message arrows, so they should be as few as possible.
SD7	Minimize longest message arrow length	The longest message arrow should be shortened if possible, e.g., placing elements closer.
SD8	Uniform message arrow length	Message arrows with similar length make diagram more understandable. Similar arrow length is also needed to fulfill the "slidability" criteria.
SD9	Improve "slidability"	"Slidability" is an aesthetic criteria for better clearness, particularly important in bigger sequence diagrams, where the whole diagram fails to fit in one screen.

A sequence diagram is specific in its visual presentation. All the objects are allocated horizontally at the top of the diagram and the life lines are drawn vertically top-down. Therefore, the criteria for the UML sequence diagram should be carefully selected or even modified, so that they could be applied. For example, one specific criterion for sequence diagram is correct sequence of messages, which is the meaning of this diagram. Poranen et al. [10] and Wong and Sun [6] have identified the criteria specific for sequence diagrams.

B. Basic Principles of the Layout Algorithm

Considering the specificity of sequence diagram the authors propose to use an algorithm, which is based on topology-shape-metrics planarization step and uses one principle of force-directed approach – object tends to attract those objects, with which it communicates. The algorithm places the elements possibly close and tries to arrange communicating participants beside based on priorities. Priorities are calculated considering object attraction forces – as more messages between elements as higher priority for them to be beside. The layout algorithm calculates the distance between the elements considering lengths of messages and class object names. Algorithm places elements as close as possible by taking into account the diagram flow (e.g., interacting objects are being placed beside if possible). The pseudo-code of the layout algorithm implemented is the following [11]:

```
func layout_SD(Sequence_object obj[], Message msg[],
Sequence_object result[])
  //determines the first object
  for i=0 to obj.size do
    if msg[i].sequenceNum = 1
       add to result object sequence msg[i].sender
  //determines priorities of other objects
  for i=0 to obj.size do
     for j=0 to msg.size do
        if (msg[i].sender = result[result.size-1] AND
        msg[i].receiver!= result[result.size-1])
            for k=0 to obj.size do
                if msg[j].receiver = obj[k]
                  obj[k].priority+1
        if (msg[i].receiver = result[result.size-1] AND
        msg[i].sender != result[result.size-1])
            for k=0 to obj.size do
                if msg[j].sender = obj[k]
                  obj[k].priority+1
     //excludes from not ordered obj[] element, that
     //was last added to result[]
     for j=0 to obj.size do
        if (obj[j]= result[result.size-1])
           delete objekts[j]
     //adds highest priority object to result[]
     for j=0 to obj.size do
        if (obj[j].priority=highest priority
          adds to result[] obj[j]
  //adds coordinates to objects
  lastXcoordinate=X_START_COORDINATE
  for i=0 to result.size do
     result[j].y=Y_ START_COORDINATE
     //MINIMAL_SPACING needs to be computed considering
     //last added object size, current object size and
     //maximal between those two object sent and received
     //message label length
     result[j].x= lastXcoordinate + MINIMAL_SPACING
     lastXcoordinate = result[j].x
```

Figure 1 shows an example of "bad" layout of the UML sequence diagram and the corresponding good layout of the same object interaction. The algorithm is implemented in BrainTool, which serves for generating UML diagrams from the two-hemisphere model mentioned above.

 ISBN: 978-1-61208-367-4

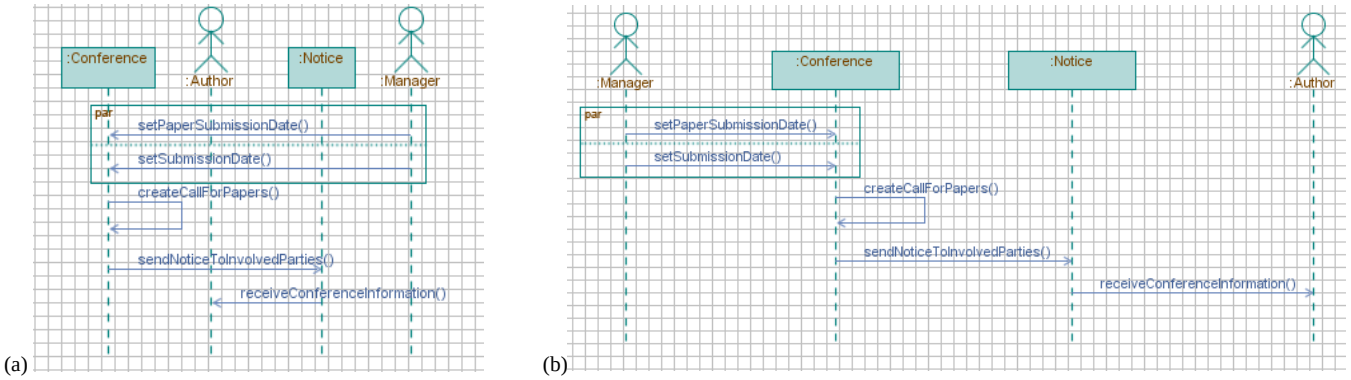


Figure 1. Example of "bad" (a) and "good" (b) layout of the UML sequence diagram.

III. LAYOUT ALGORITHM FOR THE UML CLASS DIAGRAM

The UML class diagram describes system's structure by showing its classes with the methods and attributes, and relations between classes. The visual presentation of the UML class diagram lookes like graph with vertexes and edges, but due to class diagrams ability to present different types of relationships between classes, the diagram is classified as a graph with specific constructions of arcs of different types.

A. *Layout Requirements for the UML Class Diagram*

There is no set standard for the location of classes. It is generally agreed upon to place the most important objects at top left and read the diagram to the right and downwards [6], however, by not following this rule would not make the diagram less readable.

By analogy with the UML sequence diagram, layout criteria for the UML class diagram result from the theory of perception. The same as for sequence diagram, the layout algorithm for class diagram should take into consideration all the Gestalt theory principles described above.

In addition, it is possible to define requirements' set for UML class diagram elements' layout on basis of perceptual theory. This helps to determine UML diagram's layout algorithm's tendency. Therefore, an algorithm provides the opportunity to automate layout of UML diagram's elements and transform given diagram to its normal form.

Some of defined requirements conflict with each other (for example, minimizing the subset separation requirements and exploiting the proximity requirement). This means that it is essential to define significance for conflicting requirements especially for diagrams' elements layout automation; also this ability can be given to user. We leave this task and discovering of new requirements for diagrams' elements layout for further studies.

However, it is up to the creator of the layout algorithm to decide which criterion is more important. All the described principles and requirements can be used for creation of algorithm for diagram's elements' automated layout. Table 2 shows the list of criteria for the UML class diagram layout. Criteria are marked with CD identifier.

TABLE II. CRITERIA FOR LAYOUT OF THE UML CLASS DIAGRAM

ID	Name of criterion	Description
CD0	Join inheritance arcs	Joining inheritance arcs provide a more understandable structure and suggest hierarchy. It also decreases the amount of connections to a class, which can make it easier to view.
CD1	Ensure association representation	There are several ways to represent associations, depending on how much information is shown.
CD2	Employ selectivity	Some information contained in a class or relationship can be less useful than other, so displaying only the useful information can help the understandability of the diagram.
CD3	Use colors	Many people are sensitive to colors [5]. This can be used to visually group classes.
CD4	Minimize crossings and bends	Crossings and bends can make it harder to distinguish what classes a relationship connects.
CD5	Center parents or children	Centering parents or children can visually group them together.
CD6	Reduce length of relationships	Shorter relationships help decrease the size of the diagram and make it easier to view.
CD7	Ensure inheritance direction	It is generally agreed upon, that child classes should be placed below parent class [8]. This helps display the hierarchy.
CD8	Avoid overlapping	Overlapping can cause loss of data and remove the representation of object shapes. All this leads to less readable diagrams.
CD9	Employ symmetry	Symmetry can improve the readability of a diagram.
CD10	Employ orientation	It is advisable to layout diagrams in a way, to read them from top to bottom and from left to right. This is more common in most countries and helps to guide the flow of information.
CD11	Employ orthogonality	Orthogonal relationships are easier to follow than bent or straight lines and help avoid overlapping.
CD12	Place labels horizontally	Placing all the labels horizontally helps readability of the diagram.
CD13	Place associations horizontally	Associations should be placed horizontally if possible. It helps readability of the diagram as well as placement of labels.

 ISBN: 978-1-61208-367-4

B. *Basic Principles of the Layout Algorithm*

The layout algorithm operates in four major steps [12]. Prior to these steps, algorithm gathers data on all the classes and their relationships in the diagram and places them in specific data types and constructs, for easier usage.

The pseudo-code of the layout algorithm implemented is the following [12]:

```
func layout_CD(Class_object obj[], Relationship rel[],
int iteration_count)
for n=0 to iteration_count do
   //Assign each class a score
   for i=0 to obj.size do
      set obj[i].score based on rel[obj[i]]
   //Group the classes
   while !obj[].isEmpty
      for i=0 to obj.size do
         find obj[i].score = highest
         set max=i
         set max_group_level++
      set obj[max].grouplevel
      remove obj[max] from obj[]
      for i=0 to obj.size do
         find obj[i] with rel[obj[max]]
         set obj[i].grouplevel = obj[max].grouplevel
         remove obj[i] from obj[]
   for i=0 to max_group_level
      add obj.grouplevel[i] to group[i]
   //Layout the groups
   for i=0 to group.size
      for j to group[i].obj.size
         if rel[obj[j]] is optimal do
            set obj[j].coordX AND obj[j].coordY
   //once iteration is completed, repeat,
   //assuming created groups as classes
```

During the first step, each class is assigned a score. This score is calculated based on how many relationships a class has, as well as the type of these relationships. Additionally, class content is taken into account, such as attribute and method count.

In the second step, all the classes are divided into small groups. The groups are created around the classes with the highest scores. Each group contains classes' no more than two relationships away from the main class of the group. This ensures each group is compact and contains classes with similar content. As a special condition, parent class that has generalization relationships will always be the main class of a group. All the child classes will be part of it.

The third step covers the layout of individual groups. Since the diagram is now divided into many small clusters of classes, each group contains a small and limited amount of classes. This limit can be set by the user to personalize the workflow of algorithm. Because the amount of cases is small, a simple layout can be applied, by checking the type of relationship classes have and placing them accordingly, to suit various layout criteria.

Step four involves returning and re-doing steps one to three, treating the newly created and laid out class groups as standalone classes. Because the classes are generally drawn as rectangles, so a group of classes can also be combined and displayed in a similar way. An already implemented approach that remotely resembles this is structured class notation [1].

In order to successfully implement the steps described above, a specific approach is used. All the classes and in later iterations- class groups, are placed in a container object. This object contains all the required data for the layout- class coordinates, width, height and score. Because it can contains a single class and a group of classes, the algorithm only needs to iterate trough the same object type. This improves the workflow of the algorithm.

Figure 2 shows an example of "bad" layout of the UML class diagram and the corresponding good layout of the same class structure. The class diagram consists only from seven classes, but still it is possible to demonstrate the ability of the algorithm to layout the diagram.

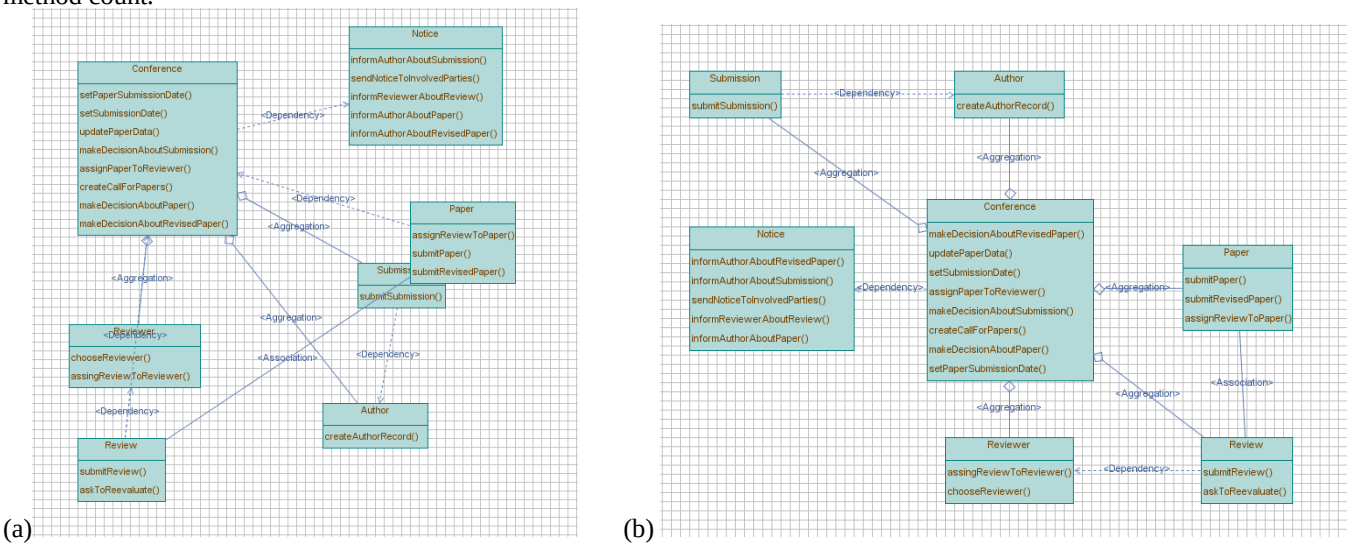


Figure 2. Example of "bad" (a) and "good" (b) layout of the UML class diagram

IV. RELATED WORK AND EVALUATION OF THE RESULT

The problem of automatic UML diagram layout still exists and it is widely discussed in relation to class and sequence diagrams. The cause of the layout problem is that algorithms are not well suited for each diagram type and there are many different aesthetic criteria to comply with. Some of the criteria are easier to implement than others, for example SD1, SD2, SD4-SD8 [10]. Another problem in automatic layout is that many of the aesthetic criteria are conflicting, e.g., message arrow length minimization (SD5) and minimization of crossings (SD4), and because reducing message arrow length is more likely to cause more crossings. The authors of [10] mention that the optimal layout is algorithmically complicated challenge, which is one more problem to automatic layout, for example an optimal linear layout problem is considered as NP-complete problem [13].

Since there are many different layout algorithms a solution can be found by studying different possibilities to tailor algorithm to specific problem or combine several of them to get the expected results. Diagram layout algorithms are based on graph theory and graph layout algorithms [14]. Algorithms can be divided into approaches, where the most used ones are topology-shape-metrics, hierarchical, visibility, divide and conquer, force-directed approaches; they are described in [7]. Genetic algorithms can also be used in diagram layout.

Topology-shape-metrics approach is one of the most used one [15]. The approach is suitable for orthogonal graphs and it supports many different aesthetic criteria [7]. Eichelberger and Schmid [9] mention that the algorithms of this approach have been used to layout UML class diagrams and are implemented in such tools as GoVisual [16] and yWorksUML [17]. The approach has three main steps, namely, planarization, orthogonalization and compaction, which are well described by di Batista et al. [7].

Hierarchical approach, also called Sugiyama approach, is also used in UML class diagram automatic layout [18]. This approach is suitable for directed acyclic graphs - more or less hierarchic graphs, which is not the sequence diagram case.

Visibility approach is the general approach suitable for various types of graphs. It has been used in entity-relationship diagram layout by Tamassia [19]. The approach also has three main steps, as mentioned in [9] and [7]. This approach can be put in the middle between both previously described approaches. Having studied this approach more closely we can conclude that this approach is less suitable for the sequence diagram than topology-shape-metrics because of its second and third steps.

Divide and conquer approach first divides graph in parts, arranges elements and then merges these parts together [7]. Regarding to sequence diagram, this approach is only suitable to diagrams with separable subsets therefore not suitable for all kinds of sequence diagrams.

Force-directed approach is suitable for undirected graphs [7]. The force-directed approach simulates a physical system of forces, where a system tries to achieve the state of minimum energy. One of main criteria in this approach is minimization of crossings, which is not the most important criteria for sequence diagrams.

There is a wide range of genetic algorithms and they can be used for various purposes, as it was mentioned by Galapovs and Nikiforova in [20]. Genetic algorithms simulate processes from nature, like mutations crossover and selection. Genetic algorithms were used in [21] for class diagram layout and according to the research results these algorithms are time consuming (20 minutes for 17 class layout).

Authors compared the relevance of each algorithm for the sequence diagram to genetic algorithms, topology-shape-metrics and force-directed approach algorithms proved to be theoretically most suitable according to how they meet the sequence diagram criteria. Other approaches are not considered to be suitable at all because they either do not consider the right order of the priorities of criteria or they are not suitable for such diagram/graph type (e.g., they are tailored for undirected, acyclic types of graphs, but the sequence diagram is directed and cyclic).

There are several tools that provide automatic diagram layout, e.g., Borland Together [22] supports automatic UML sequence diagram layout, but uses lawless set of layout criteria while Rational Rose [23] supports UML class, but does not support sequence diagram layout. Sparx Enterprise Architect [24] and Visual Paradigm [25] are tools that also provides automatic UML sequence diagram layout, however, it does not satisfy all the mentioned criteria of layout [11].

Table 3 shows how different criteria of the UML sequence diagram layout are supported by different algorithms and how they are implemented in UML modeling tools. The evaluation "Yes/No" means that criterion is/is not supported. The evaluation "partly" means that criterion is not supported completely, only part of the criterion is implemented. The evaluation "adjustable" means that criterion can be implemented by the algorithm.

The same evaluation for criteria supporting in different modeling tools according the layout of class diagram is shown in Table 4.

Researches also have been made on other types of UML diagrams. Eichelberger and Schmid [9] give researches on automatic layout of UML use case diagrams. Bist et al. presented an approach to draw sequence diagrams in technical documentation to ease communication between project members [26]. Poranen et al. proposed various criteria for drawing a sequence diagram based on traditional graph drawing aesthetics and the special nature of sequence diagrams [10]. Wong and Dabo give requirement set based on cognitive science for sequence and class diagrams, which can help to improve diagrams' readability [27].

The KIELER project [28] evaluated the usage of automatic layout and structure-based editing in the context of statecharts. It provided a platform for exploring layout alternatives and has been used for cognitive experiments evaluating established and novel modeling paradigms. However, it was rather limited in its scope and applicability.

 ISBN: 978-1-61208-367-4

TABLE III. CRITERIA EVALUATION FOR LAYOUT OF THE UML SEQUENCE DIAGRAM

Abbreviations used in the table are the following: TSMA – Topology-Shape-Metrics Approach, HA – Hierarchical Approach, VA – Visualization Approach, DCA – Divide and conquer approach , FDA – Force-Directed Approach, MSA – Multi-Scale Algorithms, GA – Genetic Algorithms, EA – Enterprise Architect, T – Together, VP – Visual Paradigm, BT – BrainTool, adj – adjustable.

ID	Name of criterion	TSMA	HA	VA	DCA	FDA	MSA	GA	EA	T	VP	BT
SD0	Precise sequence of messages	yes	yes	yes	yes	yes	yes	adj	partly	partly	partly	yes
SD1	Avoid object and lifeline overlapping	yes	no	no	no	no	adj	adj	yes	yes	yes	yes
SD2	Elements need to be arranged orthogonally	no	no	no	no	no	no	adj	partly	no	yes	yes
SD3	Diagram flow	yes	yes	yes	adj	yes	no	adj	no	no	no	yes
SD4	Minimize crossings	adj	adj	adj	adj	adj	adj	adj	no	partly	no	yes
SD5	Message arrow length minimization	adj	adj	adj	adj	adj	adj	adj	no	no	no	yes
SD6	Reduction of long message arrow number	adj	adj	adj	adj	adj	adj	adj	no	no	no	yes
SD7	Minimize longest message arrow length	yes	yes	no	adj	yes	no	adj	no	no	partly	yes
SD8	Uniform message arrow length	no	no	no	no	no	no	adj	no	no	no	no
SD9	Improve “slidability”	no	no	no	yes	yes	no	adj	no	no	no	no

TABLE IV. CRITERIA EVALUATION FOR LAYOUT OF THE UML CLASS DIAGRAM

Algorithms	CD1	CD2	CD3	CD4	CD5	CD6	CD7	CD8	CD9	CD10	CD11	CD12	CD 13
Sparx Enterprise Architect 11													
Ring	No	Yes	Yes	Yes	Poor	No	Medium	No	Poor	No	Yes	No	Yes
Ellipse	No	Yes	Yes	Yes	Poor	No	Medium	No	Poor	No	Yes	No	Yes
Box	No	Yes	Yes	Yes	Poor	No	Poor	No	Medium	No	Yes	No	Yes
Page	No	Yes	Yes	Yes	Very Poor	No	Medium	No	Poor	No	Yes	No	Yes
Di-graph	No	Yes	Yes	Yes	Good	No	Medium	Yes	Medium	Poor	Yes	No	Yes
Spring	No	Yes	Yes	Yes	Medium	No	Good	No	Poor	No	Yes	No	Yes
Right to left	No	Yes	Yes	Yes	Very Good	No	Medium	Yes	Good	Poor	Yes	Yes	Yes
Visual Paradigm 11													
Automatic	No	Yes	Yes	Man.	Very Good	No	Medium	No	Very Good	Medium	Yes	Yes	Yes
Hierarchical	Yes	Yes	Yes	Man.	Medium	No	Very Poor	Yes	Good	Medium	Yes	Medium	Yes
Orthogonal	No	Yes	Yes	Man.	Good	No	Medium	No	Very Good	Medium	Yes	Yes	Yes
Ring	No	Yes	Yes	Man.	Medium	No	Good	No	Good	No	Yes	No	Yes
Organic	No	Yes	Yes	Man.	Medium	No	Good	No	Good	No	Yes	No	Yes
Compact	No	Yes	Yes	Man.	Medium	No	Good	No	Good	No	Yes	No	Yes
MagicDraw 18.0 beta													
Class diagram	Yes	Yes	Yes	Yes	Good	No	Medium	Yes	Good	No	Yes	Yes	Yes
Hierarchycal	Yes	Yes	Yes	Yes	Good	No	Medium	Yes	Medium	No	Yes	Yes	Yes
Orthogonal	No	Yes	Yes	Yes	Good	No	Poor	No	Good	No	Yes	Yes	Yes
Organic	No	Yes	Yes	Yes	Medium	No	Good	No	Good	No	Yes	Yes	Yes
Circular	No	Yes	Yes	Yes	Medium	No	Poor	No	Good	No	Yes	Yes	Yes
Braintool													
Organic	No	Yes	Yes	Yes	Good	No	Good	No	Poor	Poor	Yes	No	Yes
Compact	No	Yes	Yes	Yes	Medium	No	Good	No	Medium	Medium	Yes	Good	Yes
Modular	No	Yes	Yes	Yes	Good	Yes	Good	Yes	Medium	Good	Yes	No	Yes

Purchase et al. analyzed graph layout aesthetics in UML diagrams, focusing on user preferences, and conducted empirical studies of human comprehension to validate those aesthetic criteria and rank their effect [8]. They also compared various UML notations, and suggested which notations are more understandable [29].

Since there are so many criteria, with some conflicting with each other, software engineers and tool designers are often overwhelmed and confused on choosing the appropriate algorithm to use. The result of the experiments with diagram import/ export and evaluation of their layout in several modeling tools shows that there are still problems with optimal allocation of diagrams elements. And still the problem is not solved. Therefore, we can assume that the algorithms offered and implemented in BrainTool is a step forward in the evolution of the UML diagram layout.

 ISBN: 978-1-61208-367-4

V. CONCLUSION

As mentioned in the introduction of the paper, the task of element placement during system modeling has an impact on better understanding of system model and more effective usage of them during development of the system. Nowadays, one of the leaders in system development is object oriented manner of software development and object oriented system modeling has its own way for presentation of different aspects of the system. Therefore, we focused on the problem of diagram layout creating UML diagrams, which is declared as a standard for presentation of software system model and provides a notation, which grows from analysis through design into implementation in object oriented programming languages.

As a notation of system modeling for different aspects of the system, UML introduces 14 types of diagrams, which can describe system from different points of view. However, Ambler stress yet in 2004 that the UML class, sequence and activity diagrams are considered more important than others [30]. And since that time, after 10 years, still the state of the art is not changed in the importance and popularity of the UML diagrams. Nowadays, commonly used UML diagrams in software development projects still are the UML class and sequence diagrams [31][32]. So far, we presented the layout algorithm and its application for UML class and sequence diagram and demonstrates the application of the algorithms in the model transformation tool – BrainTool.

The layout algorithms we offered for the two UML diagrams, namely, sequence and class, satisfy the most criteria stated for diagram layout, which are defined by different authors. In the case with the UML sequence diagram the algorithm support 8 criteria from 10 stated, whereas the best result is 4 criteria for other algorithms and two criteria for the modelling tool. In the case with the UML class diagram, the evaluation of the algorithm offered and implemented by BrainTool is also the same obvious.

ACKNOWLEDGMENT

The research presented in the paper is supported by Latvian Council of Science, No. 342/2012 "Development of Models and Methods Based on Distributed Artificial Intelligence, Knowledge Management and Advanced Web Technologies".

REFERENCES

[1] Unified Modeling Language: superstructure v.2.2, OMG. Available: http://www.omg.org/spec/UML/2.2/Superstructure [retrieved: August, 2014].

[2] BrainTool. Available at http://braintool.rtu.lv/ [retrieved: August, 2014]

[3] O. Nikiforova and M. Kirikova, "Two-hemisphere model drivenapproach: engineering based software development," The 16th International Conference Advanced Information Systems Engineering, A. Persson and J. Stirna, Eds. BerlinHeidelberg: Springer-Verlag, LNCS 3084, 09.2004, pp. 219-233.

[4] Nikiforova O., Kozacenko, and D. Ahilcenoka. "Two-Hemisphere Model Based Approach to Modelling of Object Interaction," Proceedings of the Eight International Conference on Software Engineering Advances, Mannaert H. et al. (Eds), IARIA ©, Venice, Italy, October 28-November 1, 2013, pp. 605-611.

[5] B.E. Goldstein, Sensation and Perception. Wadsworth, 2002.

[6] K. Wong and D. Sun "On evaluating the layout of UML diagrams for program comprehension," IWPC 2005, 13th International Workshop on Program Comprehension, May 15-16, 2005, St. Louis, Missouri, USA. IEEE Computer Society 2005, pp. 317-326.

[7] G. di Battista, P. Eades, R. Tamassia, and I. Tollis, Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall, 1999.

[8] H. C. Purchase, J-A. Allder, and D. Carrington, "Graph Layout Aesthetics in UML Diagrams: User Preferences," Journal of Graph Algorithms and Applications, vol. 6, no. 3, 2002, pp. 255-279. [Online]. Available: Universitat Trier, http://www.informatik.uni-trier.de [retrieved: August, 2014].

[9] H. Eichelberger and K. Schmid, "Guidelines on the aesthetic quality of UML class diagrams." In Information and Software Technology, vol. 51, no. 12, 2009, pp.1686-1698. [Available: ScienceDirect, http://www.sciencedirect.com. [retrieved: August, 2014].

[10] T. Poranen, E. Makinen, and J. Nummenmaa "How to Draw a Sequence Diagram," SPLST'03 Proceedings of the Eighth Symposium on Programming Languages and Software Tools, June 17-18, 2003, Kuopio, Finland. University of Kuopio, Department of Computer Science 2003, pp. 91-102.

[11] D. Ahilcenoka, Development of the Layout Algorithm of the UML Sequence Diagram, Master Thesis, Riga Technical University, 2014.

[12] D. Ungurs, Development of the Layout Algorithm of the UML Class Diagram, Master Thesis, Riga Technical University, 2014.

[13] M. Garey and D. Johnson, Computers and intractability - A Guide To The Theory Of NP- Completeness. W.H. FREEMAN AND COMPANY, 1991.

[14] K. Freivalds, U. Dogrusoz, and P. Kikusts, "Disconnected Graph Layout and the Polyomino Packing Approach," GD 2001 9th International Symposium on Graph Drawing, September 23-26, 2001, Vienna, Austria. Lecture Notes in Computer Science, Springer-Verlag, 2002, pp. 378-391.

[15] J. Sun, Automatic, Orthogonal Graph Layout, Project work, Hamburg University of Technology, 2007.

[16] Oreas optimization, research and software, GoVisual Diagram Editor. Available: http://www.oreas.com/gde_en.php. [retrieved: August, 2014]

[17] yWorks, Automatic Layout of Networks and Diagrams. Available: http://www.yworks.com/en/products_yfiles_practicalinfo_gallery.html [retrieved: August, 2014].

[18] J. Seemann, "Extending the Sugiyama Algorithm for Drawing UML Class Diagrams: Towards Automatic Layout of Object-Oriented Software Diagrams," GD '97, Graph Drawing, 5th International Symposium, September 18-20, 1997, Rome, Italy. New York: Springer Verlag, 1997, pp. 415-424.

[19] R. Tamassia, "New Layout Techniques for Entity-Relationship Diagrams," Proceedings of the Fourth International Conference on Entity-Relationship Approach October 29-30, 1985, Chicago, Illinois, USA. IEEE Computer Society and North-Holland, 1985, pp. 304-311.

[20] M. Mitchell, An Introduction to Genetic Algorithms. A Bradford Book, 1999.

[21] A. Galapovs and O. Nikiforova, "Several Issues on the Definition of Algorithm for the Layout of the UML Class Diagram," 3rd International Workshop on Model Driven Architecture and Modeling Driven Software Development (MDA & MDSD 2011) in conjinction with the 6th International Conference on Evaluation of Novel Approaches

 ISBN: 978-1-61208-367-4

to Software Engineering, June 8-11, 2011, Beijing, China. SciTePress Digital Library 2011, pp. 68-78.

[22] Borland a micro focus company, Borland Together. Available: http://www.borland.com/products/Together/. [retrieved: August, 2014].

[23] IBM, Rational Rose product family. Available: http://www-01.ibm.com/software/awdtools/developer/rose/. [retrieved: August, 2014]

[24] Visual Paradigm "Drawing activity diagrams". Available: http://www.visualparadigm.com/support/documents/vpumluserguide/94/200/6713_drawingactiv.html [retrieved: August, 2014].

[25] Sparx systems, "Enterprise Architect". Available: http://www.sparxsystems.com.au/ [retrieved: August, 2014].

[26] G. Bist, N. MacKinnon, and S. Murphy, "Sequence diagram presentation in technical documentation," SIGDOC 2004: Proceedings of the 22nd Annual International Conference on Design of Communication, NewYork, NY, USA, ACMPress, 2004, pp. 128–133.

[27] K. Wong and S. Dabo, "On evaluating the layout of UML diagrams for program comprehension," Software Quality Journal, 2006, pp. 233–259.

[28] KIELER project, the Kiel Integrated Environment for Layout Eclipse Rich Client [Online] Available http://www.informatik.uni-kiel.de/rtsys/kieler [retrieved: August, 2014]

[29] H.C. Purchase, M. McGill, L. Colpoys, and D. Carrington "Graph drawing aesthetics and the comprehension of UML class diagrams: an empirical study," CRPITS 2001: Australian Symposium on Information Visualisation, Australian Computer Society, Inc., 2001, pp. 129–137.

[30] S. W. Ambler. The Object Primer: Agile Model-Driven Development with UML 2.0. Third Edition. Cambridge, UK: Cambridge University Press, 2004. 572 p. ISBN 978-0521540186.

[31] Runtime Verification: First International Conference, RV 2010, St. Julians, Malta, November 1-4, 2010. Proceedings (Lecture Notes in Computer Science / Programming and Software Engineering), Barringer et al. (eds.), 2010.

[32] L. T. Yang, E. Syukur, and S. W. Loke Handbook on Mobile and Ubiquitous Computing: Status and Perspective, CRC Press, 2012.

 ISBN: 978-1-61208-367-4

Communication Aspects with *CommJ*: Initial Experiment Show Promising Improvements in Reusability and Maintainability

Ali Raza
Computer Science Department
Utah State University
Logan, Utah, USA
ali.raza@aggiemail.usu.edu

Jorge Edison Lascano
Computer Science Department
Universidad de las Fuerzas Armadas
ESPE
Sangolqui, Ecuador
jelascano@dcc.espe.edu.ec

Stephen Clyde
Computer Science Department
Utah State University
Logan, Utah, USA
stephen.clyde@usu.edu

***Abstract*—A 2013 ICSEA paper introduced *CommJ* as an extension to *AspectJ* for encapsulating communication-related crosscutting concerns in modular, conversation-aware aspects. This paper now presents preliminary, but encouraging results from a subsequent study that shows six different ways in which *CommJ* can improve the reusability and maintainability of applications requiring network communications. We begin by defining a reuse and maintenance quality model as an extension to an existing quality model. We then identify six hypotheses that can be measured using metrics from the quality model. Finally, to test the hypotheses, we compare implementations of different sample applications across two study groups: one for *CommJ* and another for *AspectJ*. Results from the study show improvement in the *CommJ* for all six areas addressed by the hypotheses.**

Keywords-aspect-oriented programming (AOPL); crosscutting concerns; AspectJ; software reuse and maintenance; software metrics.

I. INTRODUCTION

Aspect-oriented Software Development (AOSD) first started to appear in the literature in 1997 [4][12] as a way of reducing the scattering and tangling of code caused by crosscutting concerns [15]. Its contribution was to encapsulate the essence of crosscutting concerns into abstractions, called *aspects*. An aspect is an Abstract Data Type (ADT) with all of the same capabilities as an object class, plus a few enhancements. Specifically, it can contain *advice*, which is logic for implementing crosscutting concerns that is automatically woven into appropriate places in the base applications. The *aspects* also include *pointcuts*, which describe where and when the advice weaving takes place. More specifically, each pointcut identifies a set of *joinpoints*, which are intervals in the execution of the system and weaving can occur before, after, or around these intervals [15].

AspectJ is an Aspect-oriented Programming Language (AOPL) that extends *Java* for aspects [14]-[17]. It allows programmers to *weave* advice into joinpoints that correspond to constructor calls or executions, methods calls or executions, class attribute references, and exceptions. The problem is that *AspectJ*, like other AOPLs, does not support the weaving of advice into high-level abstraction, like Inter-Process Communication (IPC) where each conversation has an independent context. IPC are ubiquitous in today's software systems, yet they are rarely treated as first-class programming concepts. Instead, developers typically have to implement communication protocols using primitive operations, such as *connect*, *send*, *receive*, and *close*. The sequencing and timing of these primitive operations can be relatively complex.

The *CommJ* framework (Section II) extends *AspectJ* so developers can weave crosscutting concerns into IPC in a modular and reusable way, while keeping the core functionality oblivious to those concerns. Specifically, it allows programmers to view individual conversations as uniquely identifiable concepts, with its own context and weave logic into a base application that makes use of the context information for individual conversations.

Our study investigates potential changes to the reuse and maintenance to software when developers use *CommJ*. It does so by evaluating certain desirable characteristics defining a quality model (Section III) that can be measured by computable metrics (Section IV). Based on initial theoretic notions, we hypothesize that developers should see reuse and maintenance improvements relative to six desired qualities (Section V) defined by the quality model. Section VI talks about our experiment methodology, which required formal approval from Institutional Review Board (IRB) [10], selection of the sample software application, and identifying interesting crosscutting concerns that would give us good coverage. The methodology also included typically, supporting activities such as recruitment and training of the developers. After the experiment, we collected data from the code, surveys, hourly journals, and questionnaires.

From the results (Section VII) of the study, we conclude that IPC software components developed with *CommJ* were more cohesive and oblivious. They were also less scattered, coupled, complex and smaller in size than similar components programmed in *AspectJ*. These preliminary results lead us to believe that further experimentation with *CommJ* and refinement of its framework could prove to be very beneficial to a wide range of software systems.

II. HIGH-LEVEL OVERVIEW OF *COMMJ*

CommJ enables the partitioning of a complex communication problem into manageable cohesive concepts and promotes greater reuse with better maintainability. Figure 1 shows an architectural block diagram that represents relevant conceptual layers and their dependencies. The following paragraphs describe these high-level components and their dependencies. More details on the architecture,

 ISBN: 978-1-61208-367-4

design and examples are given in [1].

The lowest layer on the left is conceptual model, called the Universe Model for Communication (UMC). It is a formal description of common knowledge related to IPC. It describes, for example, the notation of a communication protocol in terms of role-specific state machines and message types. It then defines a conversation as an instance of two or more processes exchanging data according to the behavioral rules defined by a protocol. One important part of the UMC is the definition of message. Regardless of the system, every message is a uniquely identifiable thing (object) that is part of a conversation. How a system identifies messages and tracks their relationship to conversations are different, but the underlying concept is assumed to be true for systems that use IPC.

The next layer is the *Core CommJ Infrastructure*. It is an *AspectJ* library that defines message-event joinpoints and provides mechanisms to track conversations, which will hold value context information for communication aspects. A software developer that wants to use communication-related aspects simply has to include this library in the project.

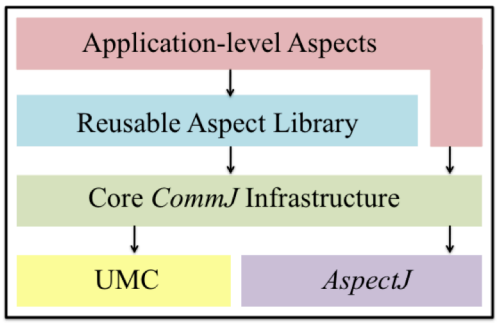


Figure 1. *CommJ* Architectural Block Diagram.

The *Reusable Aspect Library* (RAL) is a toolkit-like collection of communication aspects that application programmers should find useful for many different kinds of applications. They include aspects for measuring turn-around times, tracing conversations, and introducing behaviors into complex, multi-step protocols [1].

Application-level Aspects are those written by the application programmers, either using the abstractions provided by *CommJ* directly or by specializing the aspects in RAL. These aspects can encapsulate complex crosscutting behaviors in understandable and maintainable software components, without sacrificing obliviousness or flexibility.

III. EXTENDED QUALITY MODEL (EQM)

McCall identifies a list of eleven quality attributes [2], which have influence on quality of the software in general. Of these, we selected maintainability and reusability as the important qualities to consider initially because of potential for cost savings they both represent. Further work could focus on some of the other nine qualities.

To formalize the reuse and maintainability qualities, we adapt and extend the Sant'Anna quality model [3], because it allows for more generalized measurement, compared to Lopes' work [4] and it supports different types of implementation environments. The author builds the Quality model [3] using Basili's GQM Methodology [6]. Basili provides a three-step framework: (1) list the major goals of the empirical study, (2) derive from each goal the questions that must be answered to determine if the goals have been met; (3) decide what must be measured in order to be able to answer the questions adequately. In a nutshell, the model consists of *Qualities*, *Factors*, *Internal Attributes*, and *Metrics* (see Figures 2 and 3 for more details.).

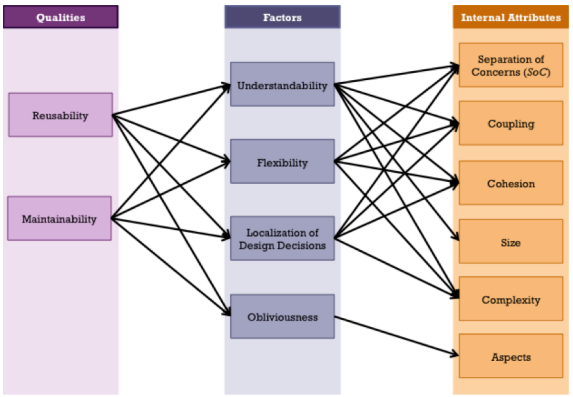


Figure 2. Extended Quality Model (*EQM*).

The qualities, such as reusability and maintainability, are the most abstract of the concepts in the model and represent the ultimate goals of "good" software. Each quality is determined by one or more factors, which are in turn determined by internal attributes. Although still abstract, these internal attributes are properties related to well-established software-engineering principles and there exists some informal notations on how to assess or evaluate them. And, that's where the metrics come in. The metrics means of measuring the internal attributes, or at least giving them a rough relative ranking. Ideally, we would like to be able to compute all metrics automatically, but that is not mandatory.

In our EQM [3], localization of design decisions, and code obliviousness were not part of original quality model [3]. However, we introduced them in our EQM for two reasons. Firstly, Parnas [27], in his landmark paper proposes three important characteristics of modular code, which were understandability, flexibility, and localization of design decisions (information hiding). Hence, reasoning maintainability and reusability only in terms of understandability and flexibility is not complete. Introduction of obliviousness is also equally important. By the time Parnas proposed the definition of modular code, obliviousness had not been invented as a fundamental design principle. However, in the context of our research experiment, which depends heavily on measuring crosscutting concerns, code obliviousness becomes very critical.

 ISBN: 978-1-61208-367-4

IV. EQM METRICS

The EQM includes 16 metrics for the six different internal attributes shown in Figure 3. Ten of the metrics can be computed automatically [20] from the code written by the subjects. The others have to be computed by hand. Below are brief descriptions of these metrics, so the reader can better understand the results presented in Section VII.

A. SoC Metrics

Separation of Concerns (SoC) defines ability to identify, encapsulate and manipulate those parts of software that are relevant to a particular concern [23]. Concern Diffusion over Application (CDA) and Concern Diffusion over Application Operations (CDO) are the two SoC metrics. CDA counts the number of primary components (class or aspect) whose main purpose is to contribute to the implementation of a concern. CDO counts the number of primary operations and advices that contribute to the implementation of a concern.

B. Coupling Metrics

Coupling is an indication of the strength of interconnections between the components in a system [24]. The EQM describes three coupling metrics. First, Coupling between Components (CBC) counts the number of other classes and aspects to which a class or an aspect is coupled. Excessive coupling of concerns increases CBC, which can be detrimental to the modular design and prevent reuse & maintenance. Depth Inheritance Tree (DIT) counts how far down in the inheritance hierarchy a class or aspect is declared. As DIT grows, the lower-level components inherit or override many methods and leads to design complexity and understanding problems. Number of Children (NOC) counts the number of children for each class or aspect. As NOC increases, the abstraction represented by the parent component can be diluted.

C. Cohesion Metrics

The cohesion of a component is a measure of the closeness of relationship between its internal components [24]. Lack of Cohesion in Operations (LCO) is the only cohesive metric in EQM that measures the cohesion of a class or aspect in our model. It does so in terms of number of method and advice pairs that do not access the same instance variable and hence should be separated.

D. Size Metrics

Size metrics physically measure the length of a software system's design and code [25]. EQM describes the following six size related metrics. Lines of Code (LOC). The greater the LOC, the more difficult it is to understand and manage the software. Method lines of Code (MLOC) is the average number of the lines of code per method. Kemerer [9] states that the greater the MLOC for a component, the more complex the component would be. Number of Operations (NO) counts the number of operations in a component. Objects with large number of operations are less likely to be reused. Number of Parameters (NP) counts the number of parameters for methods in each class or aspect. A method with more parameters is assumed to have more complex

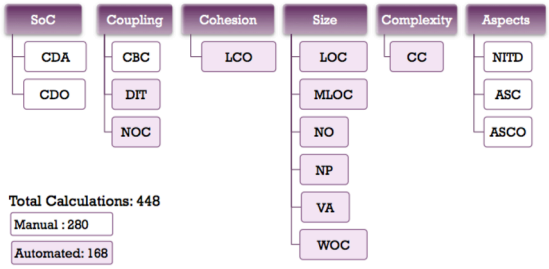


Figure 3. Measurement Metrics in *EQM*.

collaborations and may call many other method(s). Vocabulary Size (VA) counts the number of system components, i.e., the number of classes and aspects into the system. Sant'Anna [3] claims that if VA increases, it is an indication of more cohesion and less tangling for set of ADTs. Finally, Weighted Operations per Component (WOC) metric measures the complexity of a component in terms of its operations. The operation size measure is obtained by counting the number of parameters of the operation. An operation with more parameters than another is likely to be less understandable.

E. Complexity Metric

Complexity measures how components are structurally interrelated to one another. EQM uses Cyclomatic Complexity (CC) for measuring the complexity of the program. Mathematically, the cyclomatic complexity of a structured program is defined with reference to the control flow graph of the program. The metric is defined by the number of independent paths and provides an upper bound for the number of test cases that must be conducted to ensure that all statements have been executed at least once. A high value of CC affects program maintenance and reuse.

F. Obliviousness (Aspects) Metrics

Obliviousness is the idea that core functionality should not have to know about crosscutting concerns [13]. EQM defines three quality metrics for obliviousness. First, Number of Inter-type Declarations (NITD). A higher value of NITD indicates a tighter coupling between the aspect and application components. Second, Aspect Scattering over Components (ASC) counts the number of aspect components scattered over application components. It measures the tangling of aspects in the application components. More tangling of aspects in the program makes the original application less reusable and maintainable. Finally, Aspect Scattering over Component Operations (ASCO) counts the number of aspect components scattered over application component operations. ASC gives a high-level overview of the application tangling in the aspect components but ASCO provides more insight on operations-level tangling of applications inside aspect components.

V. HYPOTHESIS

The theoretical ideas that underpin *CommJ* lead to the following six hypotheses, with respect to comparing the

 ISBN: 978-1-61208-367-4

reusability and maintainability of IPC software built with *CommJ* instead of just *AspectJ*.

- *Hypothesis 1:* If crosscutting IPC concerns are effectively encapsulated in *CommJ* aspects, then the software has better *separation of concerns* and less scattering (as described by CDA, CDO in Section IV.A than equivalent systems developed with AOP design techniques.
- *Hypothesis 2:* If crosscutting IPC concerns are encapsulated in *CommJ* aspects, then the software has *lower coupling* (as described by CBC, DIT, NOC in Section IV.B) than equivalent systems developed with AOP design techniques.
- *Hypothesis 3:* If crosscutting IPC concerns are encapsulated in *CommJ* aspects, then the software has *higher cohesion* and less tangling (as described by LCO in Section IV.C) than equivalent systems developed with AOP design techniques.
- *Hypothesis 4:* If crosscutting IPC concerns are encapsulated in *CommJ* aspects, then the software is not significantly *complex* (as described by CC in Section IV.D) than equivalent systems developed with AOP design techniques.
- *Hypothesis 5:* If crosscutting IPC concerns are encapsulated in *CommJ* aspects, then the software is significantly more *oblivious* (as described by NITD, ASC, ASCO in Section IV.E) than equivalent systems developed with AOP design techniques.
- *Hypothesis 6:* If crosscutting IPC concerns are encapsulated in *CommJ* aspects, then the software is not significantly larger (as described by LOC, MLOC, NO, NP, VA, WOC in Section IV.F) than equivalent systems developed with AOP design techniques.

VI. EXPERIMENT METHODOLOGY

The research experiment consisted of the following steps:

A. Experimental Approval

In the first step, we submitted an application for conducting this Human Research Experiment to the IRB [10] and got its approval. All the researchers then passed the online human research experiment-training course offered through Collaborative Institutional Training Initiative (CITI) [11].

B. Selection of Applications and Crosscutting Concerns

We selected applications that were multithreaded, used whether JDK sockets or channels. The applications were diverse in the way they implemented IPC and therefore provide good coverage of different types of communication heterogeneities. Finally, each application supported more than one communication protocol. Table 1 lists the set of selected applications.

Since the experiment would eventually require developers to modify or extend applications for requirements that represented communication-related crosscutting concerns, our methodology included a step, which systematically selected our representative crosscutting concerns. Developers would have to apply each of these to the applications, individually. Additionally, to minimize noise in our data, we wanted to make sure that these crosscutting concerns were sufficiently simple that a novice programmer could understand the need and come up with a solution in less than 10 hours. Table 2 introduces the set of selected crosscutting concerns.

C. Recruitment and Training of Participants

To transparently recruit the candidates, we sent invitation letters and recruited seven volunteer developers who were experienced in object-oriented software development, Java and software-engineering design principles such as modularity and reusability. We then randomly organized them into two study groups: A and B. Group A programmed using an AOP approach and Group B used *CommJ*. Next, the participants completed a survey that assessed their background and skill levels. We also provided AOP training to developers in Group A, and had them worked through some practice applications. Similarly, we trained Group B

TABLE I. SELECTED SAMPLE APPLICATIONS

Application Name	Description
Levenshtein Edit-Distance Calculator (LD)	A server will calculate the LD between two input strings, provided by the client, over a connection-oriented communication.
File Transfer Program (FTP)	A file transfer protocol over connection-oriented communication.
Weather Station Simulator (WS)	A simple weather station simulator, supported by a Transmitter and a Receiver.

TABLE II. SELECTED CROSSCUTTING CONCERNS

Application Name	Description
Version Compatibility	This concern adapted one version of the message to another, so processes running different versions could still communicate with each other. The crosscutting concern included knowledge of converting one version to another and conversely
Symmetric-Key Encryption	It encrypted the communication between a sender and receiver using symmetric-key encryption
Measuring Performance	It measured some performance related statistics for message-based communications between sender and receiver

 ISBN: 978-1-61208-367-4

developers with *CommJ*, and had them worked through some practice applications.

D. Experiment Phases

In the first phase, participants filled a pre-implementation questionnaire, developed the application using initial requirements, recorded hourly journals and completed a post implementation questionnaire. In the second phase, we requested enhancements (sample applications and crosscutting concerns), had them revised their implementation accordingly, and then collected those software systems. Participants again completed the pre and post questionnaire and wrote their experiences in the hourly journals.

Finally, after the second phase, we analyzed and evaluated the reusability and maintainability using various software artifacts, which included surveys, questionnaires, hourly journals, and actual code.

We used both manual computation and automated tools to compute measurements for all 16 metrics [20]. Experiment generated a total of 28 software systems. With 16 code metrics in the EQM, we had a total of 448 measurements, 280 computed automatically with a tool [20] and 168 calculated manually.

VII. RESULTS

This section presents the data collected from the experiment and our results in context of the six hypotheses. In the following graphs, the vertical axes represent the measurements, and the horizontal axes represent the four activities of the experiment. For each activity there are two bars: a blue bar is for the results of *AspectJ* group and a green bar for *CommJ* group.

A. Hypothesis 1: Better Separation of Concerns

From the graph in Figure 4, we found that CDA and CDO values for the *CommJ* group went to zero in all four activities of the experiment. The reason for this phenomenon is that *CommJ* pointcuts provide total obliviousness between the application and communication-related crosscutting concern. *AspectJ*, components and their operations for crosscutting concern were significantly more diffused in the application because the pointcuts had to be tied to programming constructs instead of communication abstractions. From these results, we can conclude that Hypothesis 1 holds true for better separation of concerns in *CommJ* than in *AspectJ*.

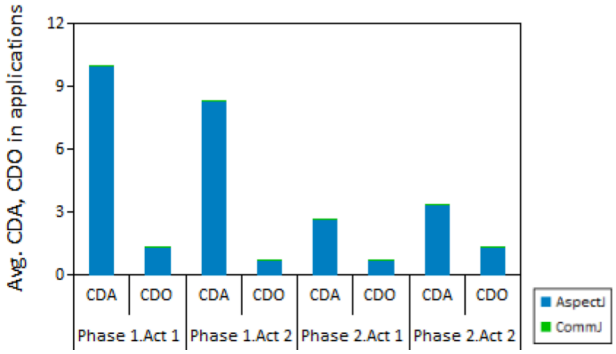


Figure 4. CDA, CDO coverage over phases.

B. Hypothesis 2: Reduced Coupling

The graph in Figure 5 indicates that *CommJ* implementations significantly reduced the values of CBC, DIT and NOC as compared to *AspectJ* implementations. *CommJ* crosscutting concerns didn't maintain any direct relationship with the application components and thus had a lower CBC value. However, in *AspectJ*, excessive coupling of concern with the application increased CBC, which hindered reuse and maintenance.

The reason for higher DIT and NOC values in *AspectJ* was that the participants preferred to override parent methods in crosscutting concerns to share data structures across aspect and application components during message passing. However, *CommJ* provides comprehensive set of pointcuts that fully encapsulates the IPC abstractions and thus participants didn't need to override or inherit the aspects.

From these results, we can conclude that Hypothesis 2 holds true for reduced coupling in *CommJ* than in *AspectJ*.

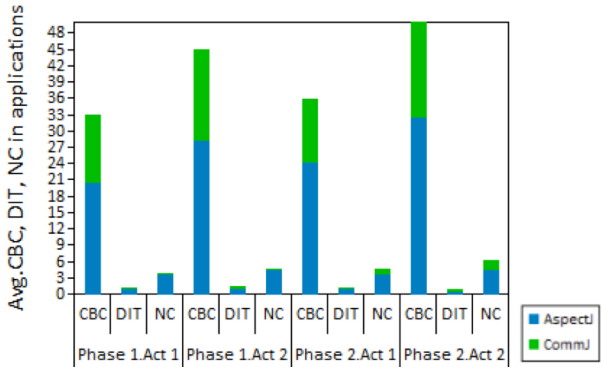


Figure 5. CBC, DIT, NC coverage over phases.

C. Hypothesis 3: Improved Cohesion

The results from the graph in Figure 6 demonstrate that *CommJ* maintains a lower value for LCOO than *AspectJ* in all phases of the experiment. Sant'Anna [3] says that LCO measures the degree to which a component implements a single logical function. Results argue that *CommJ* implementations are more cohesive and logical than *AspectJ*,

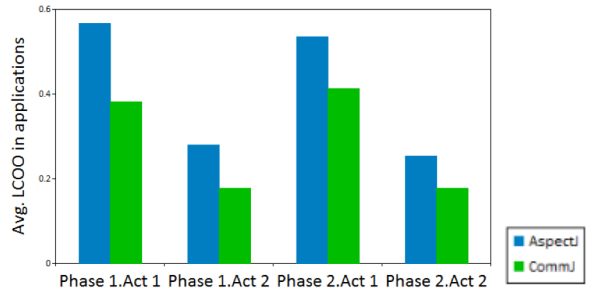


Figure 6. LCOO coverage over phases.

hence have a lower LCO value, which concludes that Hypothesis 3 holds true for increased cohesion in *CommJ* than in *AspectJ*.

D. Hypothesis 4: Reduced Complexity

The graph in Figure 7 shows that value of CC is smaller for *CommJ* than *AspectJ*, because *CommJ* hides complex IPC abstractions, which results in simple conditional statements and less tangled code. From these results, we can conclude that Hypothesis 4 holds true for less complex software in *CommJ* than *AspectJ*.

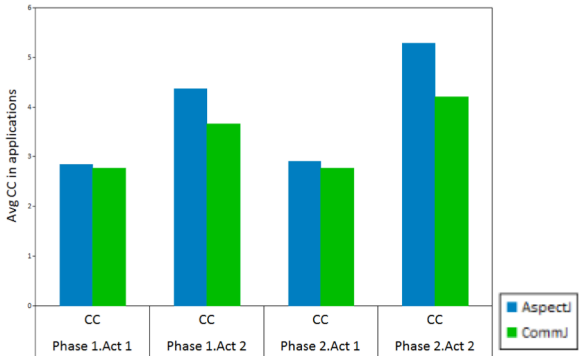


Figure 7. CC coverage over phases.

E. Hypothesis#5: Improved Obliviousness

The following graph in Figures 8 shows that *CommJ* implementations significantly reduced the values of NITD, ASC and ASCO metrics.

The reason for having a zero value for NITD in *CommJ* was that the participants used IPC constructs and did not need to use inter-type declarations (ITD) for sharing of data structures between application and aspect component. Significant reduction in ASC and ASCO was due to the layers of indirection between the application and aspect components, which *CommJ* provides but missing in *AspectJ*.

From these results, we believe that Hypothesis 5 holds true for less oblivious software concerns in *CommJ* than *AspectJ*.

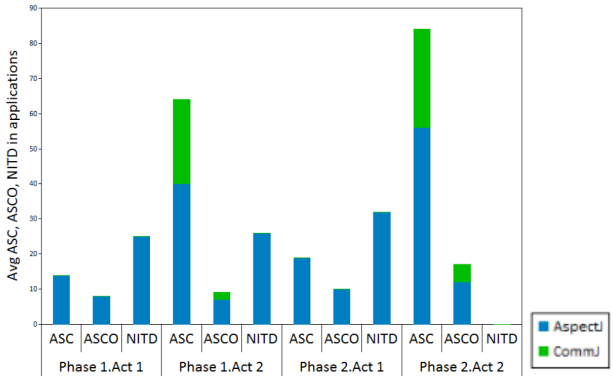


Figure 8. ASC, ASCO, NITD coverage over phases.

F. Hypothesis#6: Reduced Size

The graphs in Figure 9 shows that *CommJ* implementations significantly reduced the metrics values for LoC, MLoC, NP, NO and WOC and increase for VA in all phases of the experiment.

In comparison with *AspectJ*, *CommJ* participants found better pointcuts that helped them code the crosscutting concerns with less LOC. This is because the UMC models various general network and distributed abstractions. *CommJ* captures those abstractions in meaningful, reusable joinpoints and a family of base aspects, which helped the participants implement the application crosscutting concerns in simpler units, with no extra lines of code and fewer operations. Hence, *CommJ* reduced MLOC, NO, NP and WOC. Finally, the VA results indicate that average VA for all programs was more for *CommJ* than *AspectJ*, which, as Sant'Anna [3] claims, is an indication of more cohesion and less tangling. From these results, we can conclude that Hypothesis 6 holds true.

Besides analysis of the hypotheses via the metrics, we also collected observations through participant questionnaires and daily journals. On writing clean code, we found that 100% of *AspectJ* participants in the Phase 1 were struggled with identifying meaningful pointcuts for implementing the add-on requirements, while 33% of them still struggled with the same issue during Phase 2. On the other side, none of the *CommJ* participants struggled with this problem in either phase, which seem to indicate that *CommJ* provides simple pointcuts for IPC abstractions.

On reusability, we observed that 67% of the *AspectJ* participants in Phase 1 agreed that their applications might not run after removing the extension part from the original application. This percentage further increased to 100% in Phase 2. On the other hand, none of the *CommJ* participants felt this way for either phase. Similarly on maintainability, 100% of the *AspectJ* participants said that their changes (for either phase) introduced new dependencies in the original sample application. However, none of the *CommJ* participants felt the same way. The survey also provided some anecdotal information on frequency of bugs, specifically 67% of the participants in *AspectJ* group said that their implementation of extensions introduced new bugs in Phase 1. This percentage further increased to 100% in Phase 2. However, only 25% of the *CommJ* participants felt that their extensions introduced bugs in Phase 1 and Phase 2. This tells us that *CommJ* modularization and obliviousness may decrease the introduction of failures and the debugging time.

G. Threats to the Validity

Despite our best effort to perform the experiment objectively with minimize extraneous variables, it is important to recognize that this preliminary study has some significant threats to validity. These include variations in intelligence among the developers, health factor, work environment, and personnel commitment. Still, we believe that the results are very encouraging.

 ISBN: 978-1-61208-367-4

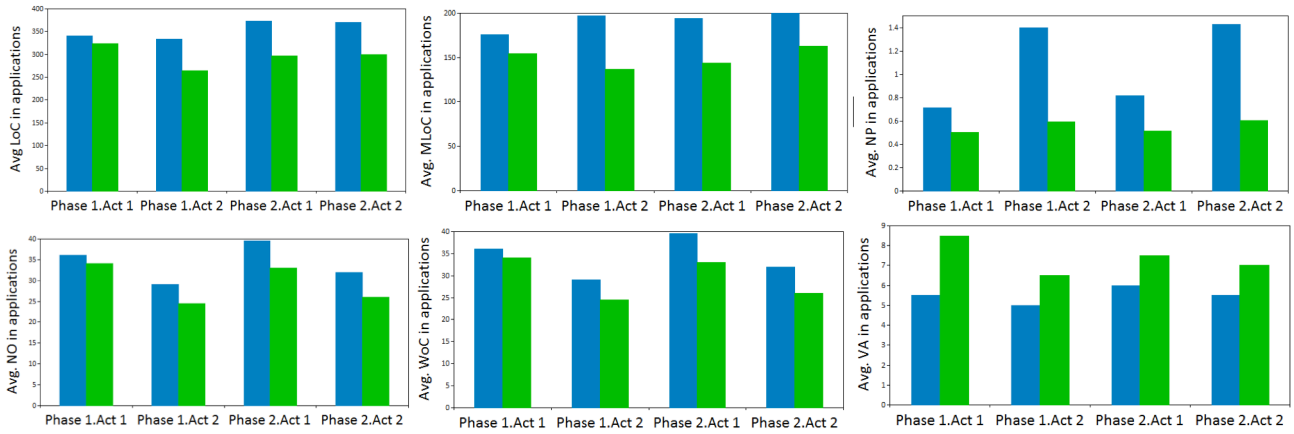


Figure 9. LoC, MLoC, NP, NO, WoC coverage over phases.

VIII. SUMMARY AND FUTURE WORK

In ICSEA 2013, we presented the design and implementation of a new AOPL framework, called *CommJ*, which allows developers to encapsulate IPC crosscutting concerns in reusable and maintainable modules [1]. This paper discusses an initial study on hoped-for benefits of *CommJ* in comparison with *AspectJ*. It defines an extended quality model, then setup an experiment methodology, involving six quality hypotheses and data collection from 28 programs. The results from this preliminary investigation provides sufficient evidence to conclude that *CommJ* is capable of encapsulating a wide range of communication-related crosscutting concerns and that it can provide better maintainability and reusability. In the future, we plan to conduct additional studies, refine the *CommJ* Infrastructure, and extend the library of reusable aspects (RAL).

REFERENCES

[1] A. Raza and S. Clyde, "Weaving Crosscutting Concerns into Inter-Process Communication (IPC) in *AspectJ,"* in ICSEA 2013, Venice, Italy, pp. 234-240.

[2] Jim A. McCall, "Factors in Software Quality," in Nat'l Tech. Information Service, 1977, vol. 1, 2 and 3.

[3] C. Sant'Anna, A. Garcia, C. Chavez, C. Lucena, and A. Von Staa, "On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework," in 17th Brazilian Symposium on Software Engineering (SEES 2003), Manaus, Brazil (2003), PUC-RioInf.MCC26/03.

[4] C. Lopes, "D: A Language Framework for Distributed Programming," in PhD Thesis, College of Computer Science, Northeastern University, 1997.

[5] J. Zhao, "Towards a Metrics Suite for Aspect-Oriented Software," in Technical-Report SE-136-25, Information Processing Society of Japan (IPSJ), March 2002.

[6] V. Basili, G. Caldiera, and H. Rombach, "The Goal Question Metric Approach," in Encyclopedia of Soft. Eng., September 1994, vol. 2, pp. 528-532, John Wiley & Sons, Inc.

[7] L. Benavides, M. Sudholt, W. Vanderperren, B. Fraine, and D. Suvee, "Explicitly distributed AOP using AWED," in AOSD 2006, pp. 51-62.

[8] G. Kiczales and M. Mezini, "Aspect-Oriented Programming and Modular Reasoning," in ICSE 2005, pp. 49-58.

[9] S. Chidamber and C. F. Kemerer, "A Metrics Suite for Object-Oriented Design," in IEEE Trans. Software Engineering, June 1994, vol. SE-20, No. 6, pp. 476–493.

[10] Institutional Review Board (IRB), http://rgs.usu.edu/irb, retrieved: August, 2014.

[11] Collaborative Institutional Trainig (CIIT), https://www.citiprogram.org, retrieved: August, 2014.

[12] G. Kiczales et al., "Aspect-oriented programming," in (ECOOP), 1997, pp. 220—242.

[13] L. Bergmans and M. Aksit, "Composing Software from Multiple Concerns: Composability and Composition Anomalies," in ICSE'2000. Position paper.

[14] AspectWorkz2, http://aspectwerkz.codehaus.org, retrieved: August, 2014.

[15] ApectJ, http://www.eclipse.org/*AspectJ*, retrieved: August, 2014.

[16] JBoss AOP, http://www.jboss.org/jbossaop, retrieved: August, 2014.

[17] Spring AOP,org.springframework, retrieved: August, 2014.

[18] C. Clifton and T. Leavens, "Obliviousness, Modular Reasoning, and the Behavior Subtyping Analogy," in SPLAT 2003.

[19] R.D. Tennent, "The Denotational Semantics of Programming Languages," in Communications of ACM 1976, pp. 437-453.

[20] Metrics plugin, http://metrics2.sourceforge.net, retrieved: August, 2014.

[21] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, "Distributed Systems: Concepts and Design (5th ed.)," in 2011, Addison-Wesley Publishing Company, USA.

[22] R. Dromey, "A Model for Software Product Quality," in IEEE Transactions on Software Engineering," in February 1995, vol. 21, No. 2, pp. 146-162.

[23] P. Tarr and S. Sutton, "N-Degrees of Separation: Multi-Dimensional Separation of Concerns," in 21st International Conference on Software Engineering, May 1999, pp. 107-119.

[24] I. Sommerville, "Software Engineering", 6th Edition, Harlow, England. Addison-Wesley. 2001.

[25] N. Fenton and S. Pfleeger, "Software Metrics: ARigorous and Practical Approach," in 2.ed. London: PWS. 1997.

[26] A. Raza,. "Improving reuse and maintenance of communication softwares with conversation-aware aspects," in Ph.D. Disseration, Computer Science Department, Utah State Univeristy 2014.

[27] D. Parnas, "On the criteria to be used in decomposing systems into modules," in Communications of the ACM 15, val. 12 (December 1972), pp. 1053-1058.

 ISBN: 978-1-61208-367-4

Customized Choreography and Requirement Template Models as a Means for Addressing Software Architects' Challenges

Nebojša Taušan, Sanja Aaramaa, Pasi Kuvaja,
Jouni Markkula, Markku Oivo
Department of Information Processing Sciences
University of Oulu
Oulu, Finland
{nebojsa.tausan, sanja.aaramaa, pasi.kuvaja,
jouni.markkula, markku.oivo}@oulu.fi

Jari Lehto
Segment Manager
Nokia Networks
Espoo, Finland
jari.lehto@nsn.com

***Abstract*—Software architecture designs are useful artifacts; however, their development and maintenance are considered challenging. To better understand the possible causes for these challenges, this article presents a case-study intended to discover and understand software architects' challenges and to propose domain-specific models to address these challenges. The main results of the case-study include a) the classification of challenges in software architecture design as well as an interpretation of the rationale behind these challenges, and b) two domain-specific models for addressing architects' challenges through architectural design. The proposed models are expected to facilitate communication between development teams, and to improve the technical aspects of the information content of requirements.**

Keywords- Software Architecture; Case-study; Choreography; Requirements Engineering; Challenge.

I. INTRODUCTION

Throughout the software product life cycle, well-established Software Architecture (SA) design is considered a valuable asset that can guarantee several quality aspects, as well as efficient development and maintenance work [1]. Today, software architects have a substantial amount of knowledge and a plethora of methods and tools at their disposable; still, well-established SA designs are scarce. One of the reasons for this situation is that, according to Falessi et al. [2], there is no SA design methodology that can simultaneously meet all the needs of an architect. In this study, the assumption is that the growing complexities of SA design challenges are one of the main reasons for the scarceness of well-established SA designs. The plethora of challenges that architects face during their work is reported in several empirical studies. Some of these studies are presented in more detail in the following paragraphs.

Smolander and Päivärinta [3] analyzed stakeholders participating in SA design and reported their problems in relation to SA. The problems, or challenges, that were expressed by software architects included: a) the continuous lack of skilled architects, which resulted in a need for well-documented SA specifications, and b) the communication mismatch, which results from architects' need to communicate with other stakeholders who often lack the necessary technical knowledge and insights.

In [4], Bosch presents his view on SA design challenges along with proposals for how to overcome them. These challenges include the lack of first-class representation, cross-cutting and intertwined design decisions, high costs of change, design rules and constraints violations, and obsolete design decisions failing to be removed from SA designs.

The challenge of enriching existing software development practices with architectural thinking is reported by Lattanze in [5]. Besides the conclusion that common methods of disseminating architectural knowledge do not work, the author proposes a list of challenges that lead to challenge state. Among others, the list includes the lack of resources for SA design, the ill-treatment of architecture activities, lack of career path for architects, and the fact that created SA designs are not used.

One of the promising ways to overcome architects' development challenges is the utilization of a Model-Driven Engineering approach [6]. In short, this approach allows architects to identify the areas in SA design that they see as particularly challenging and express these areas with Domain-Specific Models (DSM). The identified areas are then specified and managed using the concepts, rules and relationships defined in the DSM. The utilization of the domain-specific approach for the specifications and management is expected to yield several benefits, such as better comprehension of specifications, faster development and enhanced productivity [7][8][9]. The Model-Driven Engineering approach represents the overall context of this study.

To better understand and learn about SA design challenges in a real-life setting, a case-study with four software development companies was conducted. The main results are presented in this article. The main study goals were to identify a software architect's challenges and to propose DSMs as a means to address those challenges. Stated goals were reached by answering to the following research questions:

- *RQ1: What challenges do software architects face during the development and maintenance of software architecture design?*
- *RQ2: How to address the identified challenges with domain-specific models?*

These research questions were answered by conducting and analyzing five interviews with software architects,

 ISBN: 978-1-61208-367-4

analyzing additional interviews from previous studies, consulting the relevant literature, analyzing company-specific documentation, and closely collaborating with industry experts.

The stated case-study goals are also aligned with the goals of the AMALTHEA project. AMALTHEA is a European ITEA2 project of which this study is a part of, and its main goals include the development of an open source tool integration platform, the creation of an engineering methodology, and the specification of a tool-chain that will support all relevant software development areas with methods and DSMs [10]. The case-study results support AMALTHEA's goals by identifying the challenges faced by software architects on the basis of which the DSMs will be proposed. Proposed DSMs will serve as a foundation for the development of distinct tools which will become a part of AMALTHEA tool-chain.

The structure of this article consists of six sections. The following section, Section II, introduces the research method. This section is followed by the research results, which are described in Section III and Section IV. A validity discussion is presented in Section V. Concluding remarks and future research directions are outlined in Section VI.

II. RESEARCH METHOD

In this study, software architects, their challenges and model proposals are studied in their natural context. Accordingly, the case-study approach was selected as an overall research approach [11]. The research activities within the case-study were divided into two major phases, each of which sought to provide the answer to one research question. In the first phase, the SA design challenges were identified, categorized and interpreted based on knowledge gathered through an interview of the company experts. In the second phase, new DSMs were developed in such a way as to address the identified challenges. The knowledge resulting from the first phase represented the inputs to the activities in second phase. The two phases of the case-study, labeled as Phases A and B, together with the corresponding topics under investigation, the relationships between those topics, and RQs they answer, are presented in Figure 1. The research activities undertaken in these phases are described in more detail in the subsections bellow.

A. Research Phase A

The main purpose of Phase A was to provide the knowledge necessary for the development of DSM proposals. Since the DSMs seek to address the challenges faced by architects, the knowledge here implies concrete challenges, which were categorized and interpreted. For this purpose, the researchers adapted the thematic analysis method following Miles and Huberman's guidelines [12]. The main reason that a qualitative method was selected for this phase is that such a method provides a useful starting point for studying phenomena for which existing knowledge is scarce [13]. SA design challenges can be seen as such a phenomenon. The adaption of the thematic analysis will be presented through the two major phases: data collection and data analysis.

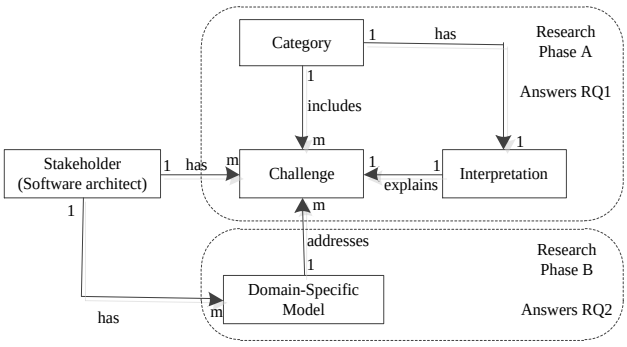


Figure 1. Case study overview.

Data collection: According to Falessi et al. [14], empirical methods, such as interviews, are suitable data collection techniques for studying SA. Following this recommendation, the authors used five interviews as the primary source of information for this study. The interviews were conducted during the first quarter of 2012, with interviewees who were working in the role of a software architect, and who had between 10 and 26 years of experience in software development.

The interviews were conducted as semi-structured, which allowed researchers to define the themes of interest, but also allowed interviewees to express their views regarding these themes in the way that was most suitable for them. Broad themes covered by the interview questions included interviewees' backgrounds, their understanding of what SA is, things that are seen as challenges and things that are seen as improvements. Additional data about the interviews are included in Table I.

In addition to the interview data, the large ICT company with which the authors collaborated provided company-specific documentation related to technical analysis. This documentation included: templates, process and work descriptions, example requirements and test specifications. This documentation was mostly used in Phase B, during the development of models, but it was also used as a means to better understand the interview response and to put these responses in context. For the purpose of data triangulation, supplementary interviews from a previous study [15] were utilized as well. Relevant information about interviewees from these supplementary interviews is presented in Table I.

To ensure the accuracy and the high quality of the data, the following measures were taken: a) The questionnaire used for the data collection was developed by a single researcher, but reviewed by at least two senior researchers and one industry expert. This was also the case for the supplementary interviews used during the study. b) The interviews were recorded, transcribed, and sent to

TABLE I. INTERVIEW DATA

Company	Type	Country	Method	Duration
A	Large ICT	A	Telephone call	1 h
A	Large ICT	A	Face to face	1.5 h
B	SME ICT	A	Telephone call	1 h
C	SME ICT	B	Telephone call	1.5 h
D	Consultant	A	Face to face	2 h
Supplementary interviews				
A	Large ICT	C	Telephone call	1 h
A	Large ICT	D	Telephone call	1.5 h

 ISBN: 978-1-61208-367-4

the interviewees for verification and for the clarification of terms that were unclear to the researchers. Upon finalization of the analysis, the results were sent for verification to industry experts in the form of technical reports and were presented in the workshops. c) Researchers worked under non-disclosure agreements and the project consortium agreement, which protected the privacy of the interviewees.

Data analysis: To aid the analysis, interview transcripts and company-specific data were imported into the NVivo tool [16], which is a software package for qualitative data analysis. A distinctive feature of this tool allowed the researchers to work on the same data sources and to continuously have insight into one another's work. This feature was especially useful because it allowed for mutual verification of work "on the fly".

At the core of the thematic analysis approach is the technique of coding. Coding allows a researcher to relate pieces of text that are of interest to the analysis with specific names or *codes*. The subsequent analysis of the text under each code facilitates the development of themes (i.e., categories) and for the rendering of interpretations. Code and category development, as well as their interpretations, are used to structure the explanation of the data analysis.

Code development: First, every piece of text that interviewees explicitly mentioned as a challenge, as well as text, that based on the researchers' expertise was known to be a challenging aspect of SA design, was encoded. The pieces of text under each code helped researchers gain a deeper understanding of SA-related problems and to formulate these problems as the challenges presented in this article. These challenges are the foundational concept of this study since they represent the basis for the development of DSMs (cf. Figure 1).

Category development: Newly formulated challenges were expressed as new codes. In the following iteration, the interview transcripts were re-coded using these new codes. The coded text was further analyzed to find commonalities, and in this case, four themes reflecting the underlying causes for the identified challenges were proposed. These themes, or categories, were used to organize the challenges and to facilitate their interpretation.

Interpretation: The final step in the data analysis was interpretation, in which the researchers combined and summarized what had been learned from the interviews with their own existing knowledge and experience. The main goal of this step was to go beyond the challenges and categories, to add the explanations and rationales behind these challenges.

The challenges, categories, interpretations, and relationships between them are illustrated in Figure 1, and, together, they represent the core knowledge necessary for the development of DSM proposals.

B. Research Phase B

Research Phase B used the results from the previous research phase for the development of DSM proposals. For this purpose, a number of workshops were organized in which industry experts, together with researchers, analyzed the challenges, categories, and their interpretations. During these workshops, challenging areas for which DSMs could be developed were identified. The first such area was described as the lack of system-level agreement on responsibilities during the implementation phase, while the second area was identified as the lack of adequate technical information in the requirement document.

Once these areas were identified, the researchers consulted the relevant literature and used company-specific materials and their own expertise to structure proposals for addressing the challenges through DSM. For the first identified area, a choreography-based DSM was proposed, while, for the second, researchers proposed a DSM for the dynamic requirement template. These two proposals were developed for the context of the case company which develops large embedded software systems and, therefore, were strongly influenced by the case company's practice. Still, the ideas within proposals are considered generic enough to be useful to architects in other companies as well.

The way in which the developed DSMs relate to the previous research phase is illustrated in Figure 1, while the more elaborate explanations of research results (i.e., challenges, categories, interpretations, and DSMs), are presented in the following two sections.

III. SOFTWARE ARCHITECTS' CHALLENGES

In this section, the results of the research Phase A are presented. These results were obtained using interview data and the thematic analysis approach, and they include the identified challenges, categories, and interpretations. Here, the derived categories are used to organize the presentation of concrete challenges and their corresponding interpretations.

A. Challenges, categorization and interpretation

The identified challenges are organized into four categories: knowledge, global software development, system size and complexity, and architectural viewpoints. This categorization seeks to reflect the underlying causes for the identified challenges.

Knowledge category: The development of SA designs, or architecting, is a knowledge-intensive process. Large amounts of both theoretical and practical knowledge are required to fulfill daily tasks. The analysis of the collected data revealed five challenges whose causes can be traced to the lack of knowledge. These challenges are summarized in Table II, and their interpretation is presented in the text below.

TABLE II. KNOWLEDGE RELATED CHALLENGES

ID	Challenge
K1	Architecting is usually experience based, without any clear statement about the rationales for design constructs or decisions.
K2	Architecting is done in the uncertain conditions. Needed information is missing.
K3	Architecting is done in the uncertain conditions Needed information is not reliable.
K4	Software architect replacement.
K5	Communicating the architecture between the developers.

K1: SA theory and SA design techniques are not sufficiently included in the educational background of software architects. Consequently, each architect devises his or her own personal understanding about SA concepts and practices and uses this understanding to specify the underlying logic behind SA designs. Since these design specifications are heavily burdened with architects' personal experiences and understandings, communicating designs to other architects becomes a challenge.

K2: Architects often do not receive the information necessary for their work. This leads to additional time consumption for information gathering and the usage of informal communication channels. What is discussed and agreed during informal communication can be important for understanding certain architectural solutions, but it often remains undocumented and can be forgotten.

K3: Two explanations for this challenge are possible: a) differences in education and experience can cause misunderstandings, and b) large systems are often documented from specific points of view. What is meaningful from one viewpoint can be irrelevant from another.

K4: During their work, architects gain knowledge about systems, interdependencies, processes, people, and customers, and they use this knowledge to develop SA designs. In some cases, architects are displaced during the course of development. The work done by a displaced architect is often poorly documented and experience based (see also K1), and for these reasons it takes a significant amount of time to train the novice architect who will continue the work of the outgoing architect.

K5: Employees often have different understandings about the same concepts. Terms like component, domain, and functional area are defined in the literature, but they are often interpreted differently by practitioners or used differently in different contexts. Refer also to challenges K1 and K3.

Global software development category: Software development companies often operate across several locations worldwide. In such a development setting, project teams are formed with developers coming from various cultural backgrounds and time zones and who communicate using non-native languages. Our analysis revealed four problems that can be linked to such a development setting (cf. Table III).

G1: Two explanations for this challenge are possible: a) For most team members, working in global development setting means communicating in a non-native language. Communicating complex issues requires a high level of language proficiency, which does not always exist. b) Global communication is done via different tools, such as emails, faxes, Wikis and voice calls. These means are not necessarily considered good substitutes for face-to-face communication.

G2: Due to mergers and acquisitions, companies are faced with the task of imposing different rules and practices. For example, if one company uses agile development, while another uses a traditional development approach, employees will be obligated to accept a new way of working.

G3: Personal acquaintances and face-to-face communication is highly appreciated among architects, and often seen as the best method of problem solving. However, this type of communication in global software development setting requires a substantial amount of resources; therefore, it always has to be justified in terms of the costs and benefits that will accompany it.

G4: Due to the variety of tasks and the large number of teams that are scattered throughout the globe, the precise responsibilities of architects are not always clear.

System size and complexity category: The interviewees work with software systems that are considered large and complex. The phrase "large and complex" emphasizes the variety of different implementation technologies, software platforms, development teams and features that such systems support. Size and complexity cause a number of challenges. The interview analysis revealed six of these challenges, which are presented in Table IV.

S1: The development of an architecture for large software systems is hampered by frequent changes, such as a) changes in organization (similar to G2), b) changes in, for example, requirement and feature documents, c) changes in release content, and d) changes in technology.

S2: Different teams prefer different practices and technologies. Sometimes, these technologies are mutually exclusive, and in these circumstances architects must decide in favor of one technological solution.

S3: System functionality can often be implemented in different architectural parts. A consensus must be reached among architects regarding which functionality will be allocated to which architectural part. This is especially important in cases for which various architectural parts are also distinct sellable items. Allocating functionality in one architectural part, means making that part a more lucrative investment option for customers.

S4: Large systems have a large number of stakeholders. Each stakeholder has his or her own vision for how the system should work, which is expressed through specific requirements. Often these requirements conflict with one another, and it is up to the architects to decide how to reconcile these conflicts.

S5: Systems tend to become large, while architects tend to become focused only on distinct parts. This state results in a loss of understanding about systems "as a whole". Systems are only valuable as a "systems" - that is, as a whole. If several parts are preforming well, but other parts are creating

TABLE III. GLOBAL SOFTWARE DEVELOPMENT CHALLENGES

ID	Challenge
G1	Difficulties in communicating tasks and results.
G2	Merging different architecting practices.
G3	Lack of personal acquaintances and face-to-face communication.
G4	Architects' responsibilities are not clear.

TABLE IV. SIZE AND COMPLEXITI CHALLENGES

ID	Challenge
S1	Architecting in a changing environment.
S2	Architecting in a heterogeneous environment.
S3	Architecting in a competitive environment.
S4	Architecting in a conflicting environment.
S5	Narrowly focused architecting.
S6	Models and tools are not sufficient for current architecting needs.
S7	Architecture and implementation often (mis)align.

 ISBN: 978-1-61208-367-4

bottlenecks, the overall system's performance becomes questionable. The performance of all parts must be balanced and planned - so that the overall performance is optimized.

S6: Conventional modeling techniques and tools are not sufficient for architects' needs. For example, the model or format of a requirement can be sufficient for one group of stakeholders, but insufficient for another. Different groups, working on different problems, have different expectations for models and tools.

S7: Large systems have large architectures that must be followed by developers. However, there are no means by which to verify that, for example, the source code for the release actually follows the architecture. Since new releases tend to reuse old designs, this misalignment can result in huge losses in time and resources.

Architectural viewpoints category: Viewpoints represent one of the crucial concepts for documenting software architecture. Architecture is actually expressed as a collection of views [17][18] based on several viewpoints. Each viewpoint emphasizes elements, and provides data that are significant only for specific concern(s) tied to a particular viewpoint. Other elements and data are omitted for clarity reasons. Based on their needs, architects can develop a feature viewpoint, a component viewpoint, a performance viewpoint, a maintenance viewpoint, and many others viewpoints they find useful. However, besides benefits, the existence of different viewpoints also causes challenges (cf. Table V).

V1: Each viewpoint represents a "world" for itself. It has its own purpose terminology, conceptualization and rules which must be known and understood in order to be effectively used, discussed and decided. Sometimes employees discuss things from the perspective of different viewpoints. This can lead to communication problems, which hamper the development process.

V2: A viewpoint addresses certain concern(s), but it does not exist in isolation. Typically, viewpoints rely on each other, meaning that updating one viewpoint often requires updating and validating other viewpoints as well. These relationships are often neither explicit, nor maintained.

V3: A reference architecture is an artifact whose purpose is to be shared across all development teams. It represents a common vision, or a shared mental model that sets common rules and terminology. The system described from this particular viewpoint is often seen as a reference for communication and development. The study revealed, however, that the reference architecture is not always properly maintained.

V4: Architectural designs, or views, are not used to their full potential. Often only a small portion of a design is used, while the rest of the information it offers remains neglected.

TABLE V. CHALLENGES RELATED TO VIEWPOINTS

ID	Challenge
V1	Employs are not aware of the existence of different viewpoints
V2	Relationship between viewpoints is not clearly visible
V3	No common, comprehensive reference architecture
V4	Architectural designs (views) are used too narrowly
V5	Architectural designs (views) are misused
V6	Difficulties in enforcing viewpoints

V5: Development problems are often discussed from only one viewpoint, and, as a result, wrong design decisions are made. For example, a static structure can be useful for an efficient breakdown of work, but it would be risky to use such a structure as a solution for certain other problems such as requirements breakdowns.

V6: In order to reach its full potential, a viewpoint must be used and understood by all interested stakeholders. A company that operates worldwide may encounter problems in enforcing certain viewpoints or practices related to these viewpoints throughout all of their global departments (refer also to G1).

IV. DOMAIN-SPECIFIC MODELS PROPOSAL

In this section, the results of the second research phase are presented. These results include two DSMs which were developed based on the identified challenges and which seek to address two subsets of those challenges. The structure for the presentation of the two models includes the following parts: a) context which explains the circumstances from which the challenges were identified; b) challenge area, which explains the architects' interest and identifies which of the identified challenges the model includes; c) proposal, which provides a description of the proposed DSM; and d) theory, which presents a short overview of the theoretical foundations underpinning the proposed DSM.

Both DSM proposals share a common underlying assumption, which is that there is an interrelationship between the product breakdown and the way in which development teams are organized. The logic of the "product-team breakdown" assumption is known in software development and reported in, for example, [19]. A simplified version of this logic is illustrated in Figure 2.

A system as a whole is subdivided into several logical components, which are further subdivided into more fine grained logical components. These components are mapped into real, physical software components, which are illustrated as the leaves of the hierarchy on the left side of Figure 2. Software development teams are organized following the same hierarchical structure. As illustrated, the board of architects is responsible for the high level conceptualization of the overall system, which is then operationalized by architects and their development teams. Each development team is responsible for a dedicated logical component, and its corresponding physical components. With this assumption in mind, the following subchapters present the detailed explanations of the two proposals.

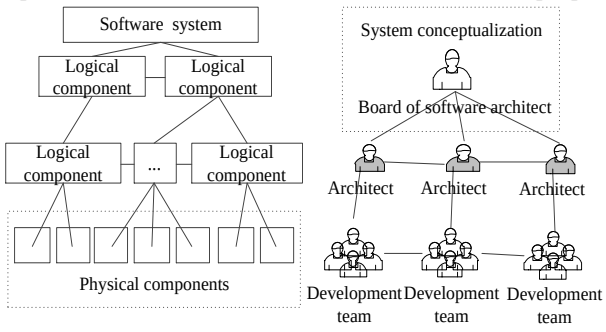


Figure 2. A system breakdown and team organization.

1) Proposal 1: Choreography based agreements

Context: The case company employs several hundred developers in its R&D division. The main task of those developers is to ensure the continuous evolution and maintenance of a large, embedded software system. The developers are organized in teams and, as illustrated in Figure 2, each team is responsible for a distinct, logical part of the system. Due to the large number of teams which are typically dispersed across different geographical, national, and cultural locations, developers are often unaware of their role in the "big picture". The "big picture" here denotes an understanding of how a developer's everyday work is aligned with the work of other teams and how it affects the functioning of a system as a whole.

Challenge area: There is no system-level agreement that would increase developers' awareness regarding who does what, and in which order. This leads to work duplication, reworks, frequent delays, and a loss of opportunities from the parallelization of work. The problem of duplication of work, for example, is explicitly stated by one of the interviewees:

"Truly, there is not such a company-level function where a decision could be made that a specific solution is implemented in a specific product and not in some other product. In practice, there may be several products that provide technical solution for system level need, and, in addition, all the solutions are standardized."

This challenge area can be seen as a collection of several of the challenges faced by architects' which have been previously identified. These challenges are K5, S3, S5, S6 and, partially, G1. An explanation of the proposal and the rationale for why it can be seen as a potential solution to these challenges is given in the text below.

Proposal: A choreography model is a way to intervene in the challenge area. The proposal is to select, customize, and provide tool support for the choreography modeling, by supplementing it with domain-specific content, and by merging it with additional models. Initial work on domain-specific supplements is begun, and some of the results are explained in Taušan et al. [20], where the way how different implementation of middleware features are affecting the choreography model is studied.

The goal behind the merger of choreography and other models is to create more ways for architects to express their designs. For example, the WS Choreography model [21] prescribes constructs for representing, e.g., the interaction. A merger provides an additional option to express interactions using techniques such as UML state charts, or UML-collaborations.

Theory: Choreography represents a system-level view of the interactions between distinct system parts [22]. The semantics of a choreography model allow architects to capture and analyze the use case in terms of participants, their roles, their messages, and the order in which those messages are exchanged in order to fulfill the use case [23]. Referring to Figure 2, each participant represents a distinct development team or engineering unit within the company. The role indicates the contribution of the architectural part, which is embodied in physical components under the team's responsibility. Messages and message ordering have to do with what is exchanged between the roles, as well as when the exchange occurs. The simplified illustration of the choreography model instance is presented in Figure 3. Here, four teams (teams x, y, z, and q) are participating in fulfilling the use-case, while the components under their responsibility take six roles (roles A, B, C, D, E and F).

The semantics of the choreography model, the experiences published in literature, and the possibilities for customizations were the main arguments for proposing it as a potential solution for the challenges in the challenge area. These arguments are discussed in more detail below.

The challenge of communicating the SA (ID: K5) is explained through the ambiguity and misunderstanding of the concepts in use. One way to address this challenge is to customize the choreography model by including domain-specific concepts. The rationale behind this approach involves reported evidence that the inclusion of domain-specific concepts can improve the comprehension and readability of specifications [7][8], which are at the core of this challenge. Moreover, this approach partially addresses the challenge of communicating tasks and results (ID: G1).

The challenge related to competing environments (ID: S3) involves allocating functionality to a set of architectural parts. Choreography natively supports the role concept for documenting the contribution that an architectural part provides to the fulfillment of the use-case. In the proposed approach, the focus is on the role, as a means of addressing this challenge, by providing the methodological and tool support for role identification and management. The rationale for using the role to understand the contribution of architectural parts at the analysis level, and to relate this to physical components during the implementation, is claimed to be a good practice by Kruger [24] and by Kruger, Nelson and Venkatesh [25].

The challenge of the narrow focus (ID: S5) involves comprehending the system as a whole and ensuring its performance. The reason choreography is seen as a suitable approach for this challenge is that it natively captures the interactions needed for the system-level use cases. As such, it imposes and documents the collaboration of all interested teams and provides insights into the roles that each team has. Regarding performance issues, the existing literature offers evidence that organizing systems according to a choreography model can result in better performance [26][27].

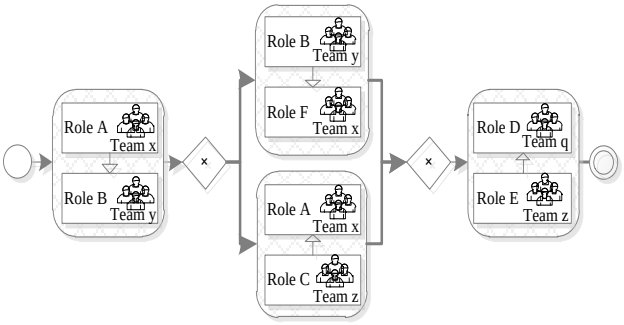


Figure 3. Choreography model.

The challenge of inadequate models (ID: S6) will be addressed through the merger of a choreography model and other models which are used by industry partners. Allowing architects to use their own preferred modeling techniques, together with the domain-specific constructs offered by the choreography model can be seen as an adequate response to this challenge.

2) Proposal 2: Dynamic requirement template

Context: In the case company, requirements are elicited by customer teams and then communicated to product management. At first, an initial screening is undertaken to quickly determine whether a requirement has the potential to bring value to the customer. If a value is identified, the requirement is analyzed in more detail from business and technical feasibility points of view (see Aaramaa et al. [15] for more details about such an analysis). This particular proposal improves the information content that is needed for the technical feasibility analysis.

Challenge area: Collecting the needed requirement information from customers and communicating this information to product management, and then to software architects, is the task of customer teams. The template for collecting and recording requirements, however, lacks the necessary technical information, and the reliability of the information in the requirement specifications is questionable. In addition, distinct technical information content has to be provided to describe each architectural part.

The direct consequence of this challenge is that architects use a significant portion of their time trying to find the necessary information, before they can begin the technical feasibility study and implementation of the requirement. This inefficient use of architects' time is only one example of the issues that are prevalent in this area, and it is also recognized by one of the interviewees:

"And because they [customer teams] *are technically not that well-trained or they don't have that kind of deep knowledge about the new functionality, (…) and then we* [software architects] *always have to make new and new inquiries towards them, to go back to the customer in order to get more information."*

This challenge area can also be seen as a collection of several architects' challenges that have previously been identified. These challenges include K2 and K3, as well as, partially, S1, S5 and S6. An explanation of our proposal and the theory that supports it is presented in the text below.

Proposal: The dynamic requirement template consists of two parts: a common and a specific part. The common part is the same for all requirements and consists of data such as the requirement's ID, name, priority, and description. The specific part is tied to a distinct part of the system or to a logical component, as is shown in Figure 2, and it consists of data that are relevant only for that specific system part. The main idea here is to use the specific part of the template to allow architects and their teams to define the information content that is relevant to their work.

This model of a requirement template is illustrated in Figure 4. The architects and their teams define the information content which includes data that have to be collected from customers, the descriptions of those data, guidance how to collect them, and criteria for the collected data's completeness. This information content forms the specific part of the requirement template. When this is done, the model is ready for instantiation by customer teams.

There are three distinct steps that can be identified during the template instantiation: a) recording data from the common part, b) understanding which parts of the systems are affected by the requirement and c) recording the specific part of the requirement for the identified system part. When these steps are completed, the requirement specification can be passed to the architects for technical feasibility analysis and implementation. It is expected that, due to the provision of focused technical data, architects and developers can do their work more efficiently.

Theory: The model behind the dynamic requirement template proposal is motivated by the idea that SA has a strong influence on Requirement Engineering (RE), and that including SA-related items in a requirement specification may result in different benefits. Some of the studies supporting this idea are presented below.

One of the first publications to focus on this idea is the panel discussion presented in Shekaran et al. [28]. In this panel, participants expressed their views on how SA is present in RE and outlined expected benefits. These benefits included an understanding of the resistance to change; the consistency, comparability, and feasibility of the requirements; and the consideration of different design alternatives.

Ferrari et al. [29] conducted a controlled experiment to understand the impact of architectures on new system requirements. The authors claimed that by considering SA during RE (among other things), analysts could elicit 10% more architecturally relevant requirements, 10% more "important" requirements, 7% more crosscutting requirements, and more implementation and interoperability requirements.

According to Cervantes et al. [30], frameworks as SA concepts influence RE. Frameworks can impose constraints such, as testability and developer skills, or create new system requirements. The example of new requirements is the case when the utilization of a concrete technology demands the usage of a concrete application type. By considering this constraint early, (i.e., in RE), losses in later development phases can be avoided.

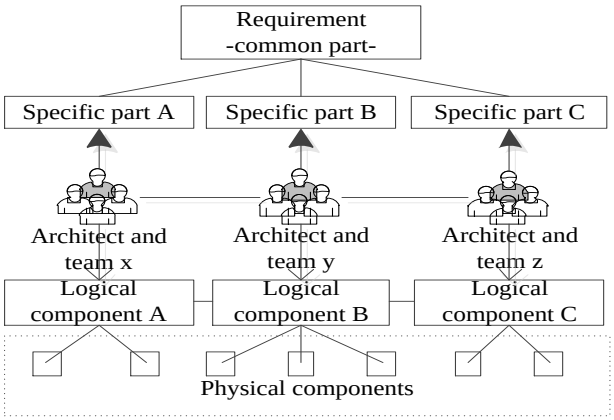


Figure 4. A dynamic requirement template.

V. VALIDITY DISCUSSION

The validity of a case-study, according to Yin [31], constitutes four aspects: construct validity, internal validity, external validity and reliability. There are several issues that may threat the validity aspects, and these were considered throughout the study.

If the researchers and the interviewees do not understand the concepts to be studied in the same way, a threat to construct validity is introduced. This threat was mitigated in this study through the rigorous peer review of the interview questionnaires that were used to collect the data for both the primary and the supplementary interviews.

The utilization of supplementary interviews can represent another threat to validity, since these data were collected for another purpose and, thus, must be considered as third-degree data [11]. Using this type of third-degree data, however, may also mitigate threats to validity, since such data's use triangulates the data; moreover in this particular study, the results of the additional interviews were in line with the primary set of data. Thus, the additional interviews addressed the validity threat to generalizability that resulted from the relatively low number of interviewees in the primary set.

The fact that the researchers have years of experience of research co-operation in the context of the case company also poses a threat to reliability in the form of researcher bias. To mitigate this threat, measures for ensuring data quality and correctness were taken. These were presented in Section II.

A threat to internal validity relates to possibilities to generalize the results and draw cause–relationship conclusions from those results. This case-study did not seek to analyze causal relationships, so, from that viewpoint internal validity has not been considered.

External validity concerns how much an analysis's results can be generalized, (i.e., used in other companies). The analysis results for this study were based on qualitative data from four companies, which develop different types of systems in different domains. The diversity of the interviewees suggests that categories and challenges could be identified in other contexts as well. The improvement proposals, however, were developed in cooperation with experts from a single company. Beyond the educated opinion that these proposals are applicable in similar type of companies or context, no other argument can be provided regarding external validity. Therefore, a threat to external validity remains.

VI. CONCLUSION AND FUTURE WORK

SA design is a solid approach to ensuring software quality and longevity. Its importance in software development is undoubtedly confirmed by one of the interviewees who, for example, claimed that:

"When we have it, [software architecture] *work comes much easier."*

The goals expressed in the AMALTHEA project, however, represent an additional empirical argument that SA design practices still need improvements. Consequently, this article presents our results from the study in AMALTHEA project which is conducted to improve the understanding of what architects perceive as challenging in their daily practice, as well as to develop ways to address these challenges with DSM.

The main results of this study are two DSM proposals. These DSMs were developed using the discovered challenges, the challenge categories (which were devised to reflect the underlying causes), and the interpretations of the challenges. In addition, existing literature, company-specific material and researcher' expertise were also used during the DSM development.

These results are also seen as answers to the research questions that where stated at the beginning of this paper. In short, based on the data analysis, RQ1 is answered by identifying, categorizing, and interpreting the architects' challenges. To answer RQ2, the researchers used the RQ1 answers and proposed two DSMs: namely, choreography-based agreements and the dynamic requirement template. These two proposals have yet to be validated. It should be also noted that, based on the identified challenges, additional DSMs could be derived as well. Which combination of challenges an architect sees as suitable for addressing through DSM is highly influenced by the architect's experience and the development context.

In addition to using these results, software architects can also recognize the derived categories and use them to predict possible challenges they will face if, for example, their company operates in a global software development setting, their product becomes large and complex, or multiple viewpoints are in use. It is also important to emphasize that the knowledge category can be seen as a pervasive category, which is present regardless of software size, complexity, the utilization of viewpoints or global software development settings. The list of challenges under each category can be seen as the concrete points that can either be addressed through an architect's choice of development practice, or serve as a means through which to raise architects' awareness about the particular challenge.

In future work, the two proposals will be fully customized to fit the case company's context. Customization will include various tasks, such as specifying of the information content for the dynamic requirement template, supplementing the choreography model with details that are relevant to the developers, and developing software support for the proposals. Additionally, the authors plan to conduct a series of evaluations with industry practitioners to validate and improve the two proposals.

ACKNOWLEDGMENTS

This study was supported by ITEA2 and TEKES. The authors would like to express their gratitude to the interviewees for their time and effort, and to the AMALTHEA partners for their cooperation. The authors are also grateful to J. Peltonen from the Tampere University of Technology for his valuable suggestions regarding the choreography model proposal.

REFERENCES

[1] L. Bass, P. Clements, and R. Kazman, Software architecture in practice. Addison-Wesley Professional, 2003.

[2] D. Falessi, G. Cantone, and P. Kruchten, “Do architecture design methods meet architects’ needs?,” The Working IEEE/IFIP Conference on Software Architecture (WICSA’07), 2007, pp. 5.

[3] K. Smolander and T. Päivärinta, “Describing and communicating software architecture in practice: observations on stakeholders and rationale,” In Advanced Information Systems Engineering, 2002, pp. 117–133.

[4] J. Bosch, “Software architecture: The next step,” in Software architecture, 2004, pp. 194–199.

[5] A. J. Lattanze, “Infusing Architectural Thinking into Organizations.,” IEEE Software, vol. 29, no. 1, 2012, pp. 19 – 22.

[6] D. C. Schmidt, “Model-driven engineering,” Computer, IEEE Computer Society, vol. 39, no. 2, 2006, pp. 25 – 31.

[7] T. Kosar, M. Mernik, and J. C. Carver, “Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments,” Empirical Software Engineering, vol. 17, no. 3, 2012, pp. 276–304.

[8] M. Völter, “Architecture as Language,” IEEE Softw., vol. 27, no. 2, 2010, pp. 56–64.

[9] A. Van Deursen, P. Klint, and J. Visser, “Domain-Specific Languages: An Annotated Bibliography,” Sigplan Notes, vol. 35, no. 6, 2000, pp. 26–36.

[10] “AMALTHEA,” 2014. [Online]. Available: http://www.amalthea-project.org/. [Accessed: 09-May-2014].

[11] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” Empirical Software Engineering, vol. 14, no. 2, 2008, pp. 131–164.

[12] M. B. Miles and A. M. Huberman, Qualitative data analysis: An expanded sourcebook. Sage, 1994.

[13] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, “Selecting empirical methods for software engineering research,” in Guide to advanced empirical software engineering, F. Shull, J. Singer, and D. I. K. Sjøberg, Eds. Springer, 2008, pp. 285–311.

[14] D. Falessi, M. A. Babar, G. Cantone, and P. Kruchten, “Applying empirical software engineering to software architecture: challenges and lessons learned,” Empirical Software Engineering, vol. 15, no. 3, 2010, pp. 250–276.

[15] S. Aaramaa, T. Kinnunen, J. Lehto, and N. Taušan, “Managing Constant Flow of Requirements: Screening Challenges in Very Large-Scale Requirements Engineering,” Product-Focused Software Process Improvement, 2013, pp. 123–137.

[16] “NVivo 10 research software for analysis and insight,” 2014. [Online]. Available: http://qsrinternational.com/products_nvivo.aspx. [Accessed: 09-May-2014].

[17] “42010-2011 - ISO/IEC/IEEE Systems and software engineering - Architecture description,” IEEE Computer Society. 2011.

[18] P. Clements, D. Garlan, L. Bass, J. Stafford, R. Nord, J. Ivers, and R. Little, Documenting software architectures: views and beyond. Pearson Education, 2002.

[19] N. Nagappan and T. Ball, “Using Software Dependencies and Churn Metrics to Predict Field Failures: An Empirical Case Study,” First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007), 2007, pp. 364–373.

[20] N. Taušan, J. Lehto, P. Kuvaja, J. Markkula, and M. Oivo, “Comparative Influence Evaluation of Middleware Features on Choreography DSL,” The Eighth International Conference on Software Engineering Advances (ICSEA 2013) IARIA, 2013, pp. 184–193.

[21] D. Burdett and N. Kavantzas, “WS choreography model overview,” W3c Work. Draft. W3C, 2004.

[22] R. Dijkman and M. Dumas, “Service-oriented design: A multi-viewpoint approach,” International journal of cooperative information systems, vol. 13, no. 4, 2004, pp. 337–368.

[23] A. Mahfouz, L. Barroca, R. Laney, and B. Nuseibeh, “From organizational requirements to service choreography,” World Conference on Services-I, 2009, pp. 546–553.

[24] I. H. Krüger, “Service specification with MSCs and roles,” IASTED Conference on Software Engineering, 2004, pp. 42–47.

[25] I. H. Krüger, E. C. Nelson, and P. K. Venkatesh, “Service-based software development for automotive applications,” Proceedings of the CONVERGENCE 2004, 2004, pp. 0.

[26] S. Cherrier, Y. M. Ghamri-Doudane, S. Lohier, and G. Roussel, “Services collaboration in wireless sensor and actuator networks: orchestration versus choreography,” IEEE Symposium on Computers and Communications (ISCC 2012), 2012, pp. 411–418.

[27] G. B. Chafle, S. Chandra, V. Mann, and M. G. Nanda, “Decentralized orchestration of composite web services,” Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters, 2004, pp. 134–143.

[28] C. Shekaran, D. Garlan, M. Jackson, N. R. Mead, C. Potts, and H. B. Reubenstein, “The role of software architecture in requirements engineering,” Proceedings of the First International Conference on Requirements Engineering, 1994, pp. 239–245.

[29] R. Ferrari, J. A. Miller, and N. H. Madhavji, “A controlled experiment to assess the impact of system architectures on new system requirements,” Requirements Engineering, vol. 15, no. 2, 2010, pp. 215–233.

[30] H. Cervantes, P. Velasco-Elizondo, and R. Kazman, “A Principled Way to Use Frameworks in Architecture Design,” IEEE Software, vol. 30, no. 2, 2013, pp. 46–53.

[31] R. K. Yin, Case study research: Design and methods, vol. 5. Sage, 2009.

 ISBN: 978-1-61208-367-4

MDD for Smartphone Application with Smartphone Feature Specific Model and GUI Builder

Koji Matsui and Saeko Matsuura
Division of Electrical Engineering and Computer Science,
Graduate School of Engineering and Science.
{ma14097@, matsuura@se.}shibaura-it.ac.jp

***Abstract*—Unlike general PC applications, smartphone applications have three innovative features that make useful mobile services a possibility. Conventional code-centric development tools used for general PC applications are not efficient for developing high-quality software with mobile features. The difficulty with conventional development is because of the variety of platforms and operations. Model Driven Development (MDD) is a promising approach to develop high-quality software products efficiently. To develop richer applications using such features, we propose a UML-based MDD method. This method uses a Smartphone Feature Specific Model and a GUI builder, independent of any specific OSs.**

Keywords-MDD; UML; Smartphone Application; GUI builder.

I. INTRODUCTION

Smartphone applications have three innovative features that present a possibility of useful mobile services. The first feature is that the device is equipped with various types of hardware. This enables the user to input a variety of data; for example, user actions that cannot be expressed by characters. The second feature is that the application can be easily extended by connecting external applications, such as *Intent* in *Android,* or *URLScheme* in *iOS* using various communication mechanisms. The third feature is a set of rich User Interface components for multi-touch devices that enables us to use the interface to improve the operability of a smartphone.

The development of applications for a smartphone is a complicated task because of the variability of platforms and the number of different devices that need to be supported. Moreover, the basic design of a target application that includes UI operability needs to be analyzed at the early stages of development to reduce the need to rework.

GUI builder allows a developer to arrange widgets using a drag-and-drop WYSIWYG editor, so that he/she can develop the user interface of the application in an intuitive manner. However, the intuitiveness of the interface is entirely dependent on the specific programming language and the analysis of the application logic. This relationship tends to be insufficient in regards to the first two features.

We propose a unified modeling language (UML)[1]-based Model Driven Development (MDD) method using a smartphone feature-specific model and a GUI builder that is platform independent.

The remainder of the paper is organized as follows. Section II discusses how to develop smartphone applications efficiently. Section III explains how to model the smartphone application using suitable development tools stated in our approach. Then, the related work is discussed in Section IV.

II. PROBLEMS IN DEVELOPMENT OF SMARTPHONE APPLICATION

Since mobile services with the abovementioned features support varied platforms and operations, it is difficult to implement conventional code-centric development to develop such a system efficiently. MDD [2] [3] is a promising approach to develop high-quality software products efficiently because it enables code generation and has high traceability.

The issue with changeability of platforms can be solved by separating concerns about platforms. The Platform Independent Model (PIM) and the Platform Specific Model (PSM) use UML. However, to realize appropriate operability, we need to design a system that uses a concrete screen image.

A developer can use GUI builder [4] for the specified OSs and develop application user interfaces in an intuitive manner. However, the intuitiveness of the interface is entirely dependent on the specific programming language and the analysis of the application logic. This relationship tends to be insufficient. Thus, the product developed using GUI builder is difficult to reuse in other applications and cannot follow various requirements changes.

UML is a well-known general-purpose modeling language that provides a standard method to visualize the design of a system. There are several convenient UML editors, such as astah* [5]. astah* and other UML editors are effective tools to design the static structure and behavior of a system; however, these tools are unsuitable to design GUI in an intuitive manner.

The problem is how to efficiently develop smartphone applications that deliver feasible static content, as well as an intuitive behavioral model. Further, these applications must also be independent of any specific OSs.

 ISBN: 978-1-61208-367-4

III. UML-BASED MODEL DRIVEN DEVELOPMENT METHOD

A. Overview of Development Process

To solve the above-mentioned problem, we propose a UML-based MDD method using a *Smartphone Feature Specific Model* and an original *GUI builder* independent of any specific OSs. Figure 1 shows an overview of our method.

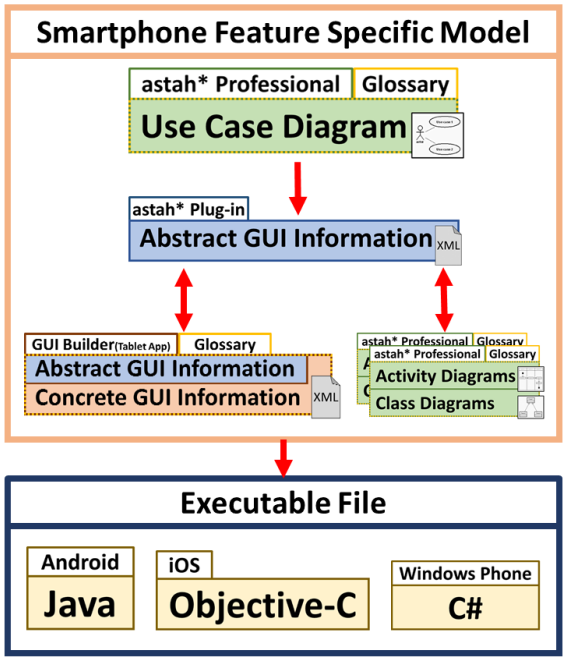


Figure 1. Overview of our Method

Use case analysis [6] is known as an effective method to define functional requirements. Therefore, because a use case represents a basic unit of function that is used by an end-user, we begin the method by constructing a use case diagram

The Smartphone Feature Specific Model consists of two types of data edited by such different design views as a UML modeling tool and the GUI builder.

The first data is a UML Model specified by a glossary of smartphone features, as shown in Figure 2. A UML Model consists of a use case diagram and a pair of an activity and a class diagram. An activity diagram and a class diagram correspond to a use case. A developer edits the pair using the UML modeling tool, astah*.

The second data is defined by the GUI builder and consists of *Abstract GUI Information* and *Concrete GUI Information.* The former consists of abstract components that are common in the Android, iOS, and Windows Phone SDK [7] [8] [9]. Moreover, the second data is connected with the UML Model by a mapping rule based on a meaning of a use case. The latter shows properties such as size, position, font, color, and concrete values, which are added to the first data.

The mapping rule defines mutual transformation between both data defined by the UML modeling tool and the GUI builder. The data of the UML Model is extracted using the astah* API Plug-in.

After a developer edits a target application using proper views that he/she thinks fit to design such aspects as function, structure, behavior, and operability. *Smartphone Feature Specific Model* data is written in XML and can be translated into codes in specific programming language such as Java, Objective-C, and C#.

B. Glossary of smartphone features.

Figure 2 shows a glossary of smartphone features mentioned in Section I. Smartphone features are classified into four classes: *View*, *Gesture*, *State*, and *ExternalSystem*. These classes are used as basic components of the smartphone specific model.

View consists of 13 *Widgets* and 5 *Layout* classes that are used for editing on the GUI builder. *Widgets* are

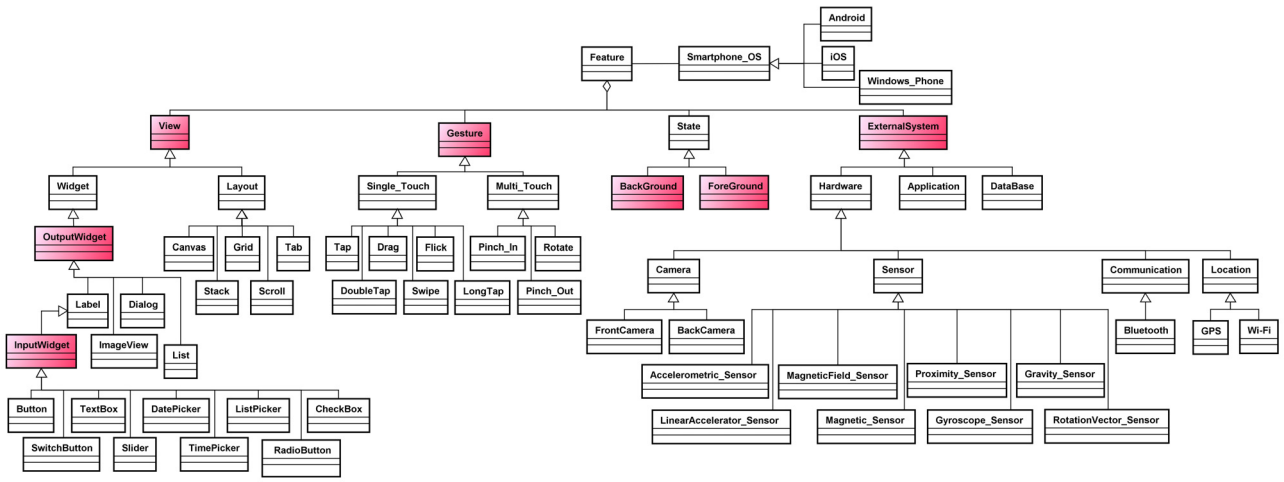


Figure 2. The Glossary of Smartphone Features

 ISBN: 978-1-61208-367-4

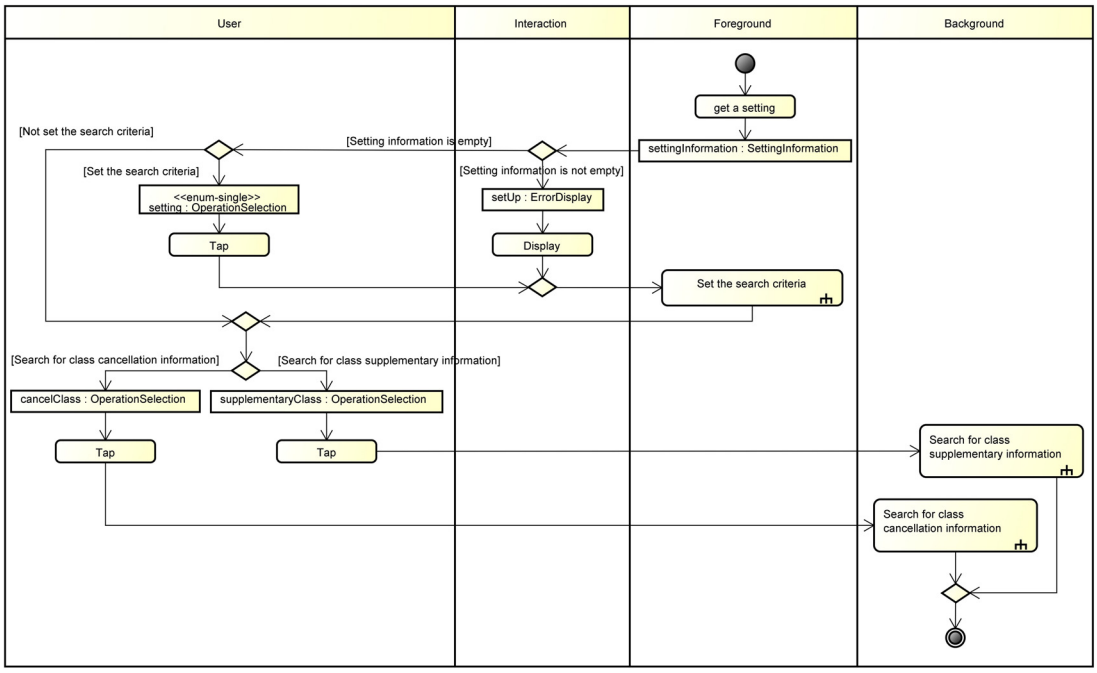


Figure 3. Activity Diagram for a Use Case

classified into *InputWidget* and *OutputWidget* and both of become components in a class diagram corresponding to a use case.

The Gesture class has a role of expressing requests for the operability of a system.

The State class is used to express a distinction of a property of a process in an early stage of development. Background processing has API usage restrictions.

The ExternalSystem class expresses a system with which an application can cooperate to improve the service. The class becomes an object node in an activity diagram and an actor of the use case. These classes include not only cooperating with other applications, but also the use of various types of hardware and communication methods.

C. UML Models

A use case diagram includes several use cases with the related actor, such as a user or available external application or hardware component. A developer may decide a root use case by relating the other use cases using *extend* or *include* relationships. The root use case represents a scenario of starting the application. By the end of the operation, each use case is defined by an activity diagram and the relationship is expressed by calling a sub activity corresponding to the other use case in the activity diagram (Figure 3).

An activity diagram expresses a series of processing actions with related data. The background action is distinguished from the foreground action by the use of a partition. An object node is used to denote the linking of external applications or hardware. A *User* partition includes user actions with input data whereas the *Interaction* partition includes actions with output data through a user interface.

Figure 4 shows a class structure of system partition specified by *State* of the glossary. In this model, the general components in an activity diagram are specified by smartphone features. A class of the glossary is displayed in red and a developer can design smartphone features by using this class.

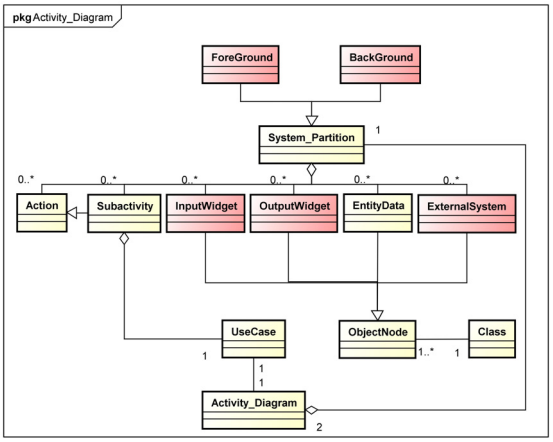


Figure 4. System Partition in Activity Diagram

Figure 5 shows how a use case corresponds to a screen of the application preventing complication of models. Information about input/output data that are used in the use case is expressed by a class diagram composed of a class

 ISBN: 978-1-61208-367-4

corresponding to three types of classes in the glossary. Entity data is also defined by a class related to the use case, as shown in Figure 5.

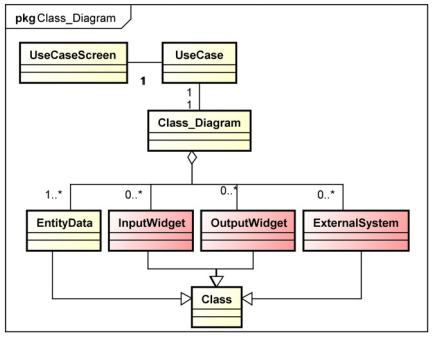


Figure 5. Use case and Class Diagram

D. UI Design with GUI Builder

Based on a use case diagram, a developer can design the UI screen image using the drag-and-drop WYSIWYG editor. In this step, the UI is designed using subclasses of *View* and *Gesture* in the glossary. As Figure 6 shows, there are 17 types of widgets in *View*. Each *View* has one or more *Gestures* that is a trigger to call the use case function. *EventAction* objects can have a connection with the other use cases. *EntityData* object expresses data that is created by the function and will be read by the *View* object. *Abstract GUI Information* is automatically generated or updated by these operations on the GUI builder. This sequence of operations corresponds to a sequence of actions in the related activity diagram.

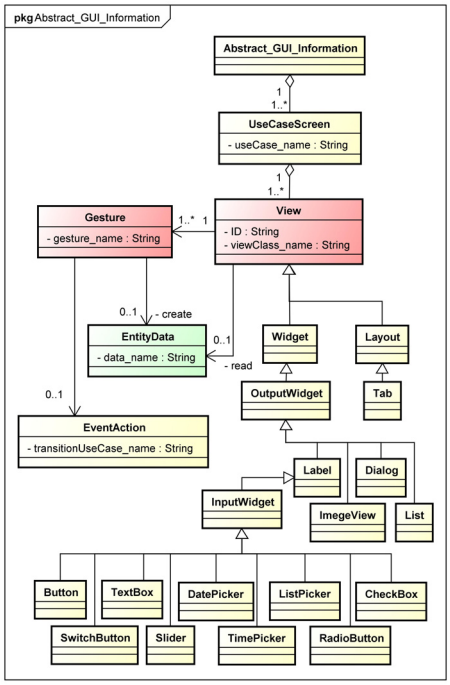


Figure 6. Abstract GUI Information

A developer defines attributes or values such as the size of a widget, the position, font type and font size, the content of the message presented, and a name of a label or button. Such data is saved as *Concrete GUI Information* written in XML.

Another important role of the GUI builder is to decide the most appropriate screen transition based on the amount of information caused by the combination of use cases. Figure 7 shows an example of how such a decision is made, to integrate two screens into one screen or not to integrate the screens.

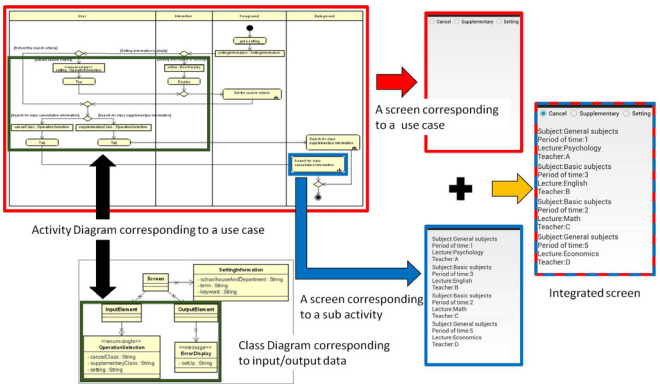


Figure 7. Products of UML-based MDD

IV. Related Work

Lettner et al. [10] has stated that MDD is a promising approach for mobile phones in solving problems of conventional code-centric development approaches. They discuss the problem from the viewpoint of the reusability of parts of a system and the adaptability to various changing platforms. However, Lettner did not propose a concrete mechanism in which we can design reusable models with smartphone specific features independent of specific OSs.

There have been several studies of MDD for smartphone applications. In one study, Sabraou, et al. [11] proposed a MDD method to design GUI using object diagrams in UML. These diagrams are translated into XML based data on the Android GUI Meta model. However, in comparison with our approach, a developer cannot design the user interface in an intuitive manner. Moreover, consideration of screen transition in accordance of the amount of information is not discussed in the Sabraou study.

In another study, Diep et al. [12] proposed an MDD environment to provide developers with a platform-independent GUI design for mobile applications. Though the static screen composition can be defined, dynamic screen changes cannot be performed. In contrast, we use a GUI builder to design GUI. Moreover, we analyze application logic called by UI components using the activity diagram and the entity data. This combination ensures that dynamic screen changes can be performed.

 ISBN: 978-1-61208-367-4

MD2 [13] is a framework for cross-platform model-driven mobile development. In their approach, a developer needs to design an application model by using a specific DSL in text form. However, the DSL is insufficient to flexibly design the smartphone application model from both the structural view and the UI view.

Franzago et al. [14] also proposed a collaborative framework for the development of data-intensive mobile applications exploiting MDD techniques and separation of concerns. Our approach uses familiar modeling Language UML and GUI builder which can easily use in intuitive manner.

V. CONCLUSION AND FUTURE WORK

In our paper, we proposed an MDD method by using an existing UML modeling tool and our own GUI builder to operate abstract widgets in an intuitive manner. This allows flexibility in the design of the smartphone application from both the structural view and the UI view. We are currently developing the GUI builder using the Android tablet PC based on the Smartphone Feature Specific Model. By applying our method to more smartphone applications, we will verify if minute differences between features of OSs can be discussed on the model.

REFERENCES

[1] OMG," Unified Modeling Language", http://www.uml.org/ (accessed: Aug. 13, 2014)

[2] S. J. Mellor, K. Scott, A. Uhl, and D. Weise, MDA Distilled Principles of Model-Driven Architecture. Addison-Wesley, 2004.

[3] OMG, "MDA Guide Version 1.0.1." Object Management Group, Tech. Rep., 2003.

[4] ADT Plugin, http://developer.android.com/tools/sdk/eclipse-adt.html (accessed: Aug. 13, 2014)

[5] astah:http://astah.change-vision.com/ja/ (accessed: Aug. 13, 2014)

[6] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. Object-oriented software engineering: A usecase driven approach, Addison-Wesley Publishing, 1992.

[7] Android developers, http://developer.android.com/index.html, 2014

[8] iOS Developer Library, https://developer.apple.com/library/ios/navigation/ (accessed: Aug. 13, 2014)

[9] Windows Phone Dev Center, https://dev.windowsphone.com/en-us/home (accessed: Aug. 13, 2014)

[10] M. Lettner and M. Tschernuth "Applied MDA for Embedded Devices: Software design and code generation for a low-cost mobile phone", the 34th Annual IEEE Computer Software and Applications Conference Workshops, 2010, pp. 63-68.

[11] A. Sabraou, M. E. Koutb, and I. Khriss, "GUI Code Generation for Android Applications Using a MDA Approach", International Conference on Complex Systems, 2012, pp. 1-6.

[12] C. K. Diep, Q. N. Tran and M. T. Tran, "Online Model-driven IDE to Design GUIs For Cross-platform Mobile Applications", SolCT, ACM International Conference Proceeding Series, 2013, pp. 294-300.

[13] H. Heitkotter, A. T. Majchrzak and K. Herbert. "Cross-platform model-driven development of mobile applications with md 2." Proceedings of the 28th Annual ACM Symposium on Applied Computing. ACM, pp. 405–411, 2013.

[14] M. Franzago, H. Muccini and I. Malavolta, Towards a collaborative framework for the design and development of data-intensive mobile applications. In Proceedings of the 1st International Conference on Mobile Software Engineering and Systems (MOBILESoft 2014). pp. 58-61, 2014.

 ISBN: 978-1-61208-367-4

Quality-Oriented Requirements Engineering for Agile Development of RESTful Participation Service

Michael Gebhart
iteratec GmbH
Stuttgart, Germany
michael.gebhart@iteratec.de

Pascal Giessler, Pascal Burkhardt, Sebastian Abeck
Cooperation & Management
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
pascal.giessler@student.kit.edu,
pascal.burkhardt@student.kit.edu, abeck@kit.edu

***Abstract*—Decision-making between humans is a recurring challenge in a society where consensuses for disagreements have to be found. To support such decision-makings, at the Karlsruhe Institute of Technology a Participation Service is developed as part of a service-oriented campus system in an agile manner and based on the Representational State Transfer (REST) paradigm. One of the key success factors of such software projects is the requirements engineering process. Scenarios are an appropriate way to describe a system from the user's point of view. However, existing methodologies do not specify quality requirements for these scenarios. This article presents an enhancement of existing scenario-based requirements engineering techniques to fulfill the quality characteristics of the international standard ISO/IEC/IEEE 29148 and align the quality aspects to the product strategy. We illustrate the approach and the resulting quality improvements by eliciting functional and non-functional requirements for the Participation Service in an agile manner, while considering constraints emerged from the existing RESTful system.**

Keywords: requirements engineering; agile; scenario; rest; service; participation; iso 29148

I. INTRODUCTION

In a society, decision-making is always a recurring and complex challenge. Several stakeholders and participants defend their points of view and try to convince the others of their personal opinion. To overcome these disagreements, consensuses have to be found that satisfy all participants.

To support the process of decision-making in a society by Information Technology (IT), at the Karlsruhe Institute of Technology (KIT) a software solution is to be created that is part of the existing service-oriented KIT Smart Campus System. This system is a collection of functionality for students for supporting their life on the campus of the university. The required software solution consists of a so-called Participation Service that provides the required functionality. The Participation Service is based on the idea of systemic consenting. This approach describes how to find a compromise or consensus that is near to an optimal consensus of the group. For that purpose, compared to usual decision-making processes, possible solutions are not scored with agreement points but with refusing points. This means, after describing the issue and collecting possible solutions, the one is selected that has the fewest refusing points. This solution represents the one with minimum resistance. As the Participation Service is required to be used by different devices, such as smartphones and tablets, it is expected to be developed as a web service based on the Representational State Transfer (REST) paradigm [1] as lightweight alternative to technologies, such as SOAP over Hypertext Transfer Protocol (HTTP), Extensible Markup Language (XML), and Web Services Description Language (WSDL). Furthermore, the service is developed in an agile manner.

One of the key success factors of such software projects is the requirements engineering process [2][3], i.e., the way how functional and non-functional requirements are captured. The usage of scenarios has evolved as an appropriate methodology to describe a system from the user's point of view. As the requirements constitute the basis for the developed software system, the quality of these requirements is very important. For that purpose, the IEEE recommended practice for software requirements specifications [4] and ISO/IEC/IEEE 29148 [5] were created. However, existing requirements engineering methodologies do not consider these quality requirements.

This article enhances existing methodologies in a way that quality characteristics of the international standard ISO/IEC/IEEE 29148 [5] are considered. For that purpose, the quality characteristics in [5] are analyzed and existing requirements engineering methodologies are described step by step and adapted when necessary.

To illustrate the approach, the resulting methodology is directly applied to the Participation Service at the KIT as a real-world project. After identification of stakeholders, the goals are elicited and prioritized. Finally, functional and non-functional requirements are formalized that fulfill the quality characteristics.

The article is organized as follows: Section II examines existing work in the context of requirements engineering methodologies. The Participation Service scenario is described in Section III. In Section IV, our methodology is presented and directly applied to the scenario by considering the quality characteristics and existing constraints. Section V concludes this article and introduces future research work.

 ISBN: 978-1-61208-367-4

II. BACKGROUND

This section analyzes existing approaches in the context of requirements engineering methodologies that identify the goals of stakeholders and writes them down in a precise way so that they can be used in the following development phases [6].

In [4], the IEEE offers an official recommended practice for software requirements specifications, which was replaced by the new international standard ISO/IEC/IEEE 29148 [5]. Based on them, quality characteristics for high quality requirements can be derived. Furthermore, the new standard provides language criteria for writing textual requirements and requirements attributes to support requirement analysis. It also provides guidance for applying requirements-related processes. These concepts will be used to analyze existing scenario-based requirements engineering methodologies and to design the one introduced in this article.

Sharp et al. [7] present a domain-independent approach for identification of the stakeholders based on four determined groups of so-called baseline stakeholders. They can be further refined in three different groups based on their role. This approach will be used to identify the stakeholders in this article. However in large projects, the resulting network of stakeholders can be huge.

For that reason, Ackermann et al. [8] describe a method with a matrix in which the stakeholders were arranged by their importance and their influence on the project. This method can be used to prioritize the discovered stakeholders for the project.

There are different requirement types, which have to be taken into account when eliciting requirements for a software product. Glinz [9] provides a concern-based taxonomy of requirements, which consists of functional requirements, non-functional requirements, and constraints. These types will be reflected in the introduced requirements engineering methodology, however with one difference: The performance will not be considered as a separate entity since it is already an ingredient of [10].

For eliciting functional requirements, Rolland et al. [11] present a goal modeling approach by using scenarios. A goal represents something that the stakeholders want to have in the future, while a scenario represents the required interactions between two actors to achieve the corresponding goal. Once a scenario has been composed, it is investigated to addict more goals. This approach can be aligned with [5], which is why it will be reused in this article.

However, there are two issues: 1) Goals cannot be regarded separately, because they could be composed of existing goals and 2) the recursive process is repeated until no more subgoals can be derived, but this can lead to a big bunch of subgoals. A solution for 1) is a repository of already analyzed goals, which can be reused by reference. The determination of a threshold in 2) is difficult, because it cannot be set easily by metrics. So the requirements engineer has to decide on its own when the abstraction meets its expectations. For this purpose, some conditions had to be found, which support the decision-making. Furthermore, it is not obvious, how to achieve the initial goals.

At this point, Bruegge and Dutoit [12] introduce some interview questions that can be used for identification of the initial goals. Furthermore, elicitation techniques can be found in [2]. To support agile software engineering, the discovered goals have to be arranged by importance to select the goals with the highest rank similar to iteration.

For that reason, the approach by Karlsson and Ryan [13] will be applied, which uses pairwise comparisons in consideration of cost and value. But, for many goals, this approach will rapidly become impracticable as the number of comparisons increases significantly. For that reason and the statement “Keep the prioritization as simple as possible to help you make the necessary development choices” by Wiegers [14], a simple classification approach with three different scales based on [4] is best suited for the initial prioritization.

When writing scenarios, the quality characteristics by [5] have to be considered. Glinz [15] presents an approach, which respects the quality characteristics by the old recommendation [4]. His findings will be used to improve the quality of requirements.

Also, Terzakis [16] presents techniques for writing higher quality requirements by providing an overview of requirements and pitfalls by using the natural language for their description. Based on this, the quality of requirements will be improved even further.

In [10], the ISO provides a quality model comprising quality characteristics that are further decomposed into sub-characteristics. This model will be used for determining the quality aspects of a software product.

For eliciting non-functional requirements, the approach by Ozkaya et al. [17] will be used. Due to the fact that statements like “The system shall be maintainable” are imprecise and not very helpful, this approach is using so-called quality attribute scenarios. Based on these, the corresponding quality characteristic of ISO 25010 [10] can be derived. However, for many quality characteristics it can be very time-consuming.

To reduce the effort, the decision-making approach by Saaty [18] will be applied by using pairwise comparison of the quality characteristics in [10] with regard to their importance for the product strategy.

With the provided constraints of the architectural style REST in [1], the last requirement type according to the taxonomy in [9] will be considered.

III. SCENARIO

To illustrate the requirements engineering approach, the KIT Smart Campus System at KIT is to be enhanced by a new service, the Participation Service. The Participation Service is designed to support the process of decision-making between professors, students, and other KIT members according to the principle of systemic consenting.

In the first phase, participants can create and describe their own subjects of debate and share them to a group of participants. In the second phase, the participants rate suggestions by expressing their dislike instead of their like as usually expected. They are able to do that in the form of refusing points from zero to ten. Refusing points indicate

 ISBN: 978-1-61208-367-4

how much a participant dislikes a possible suggestion. Thus, rating a suggestion with zero refusing points means that the participant totally agrees with the suggestion. Rating a suggestion with ten refusing points means that the participant rejects the suggestion. The suggestion with the fewest amount of refusing points represents the one with the highest acceptance of all participants. This suggestion has minimum resistance and is the consensus of the group. Fig. 1 illustrates the described process. For example, the Participation Service can be used for determining new lecture contents in collaboration with students in the context of the Research Group Cooperation & Management (C&M).

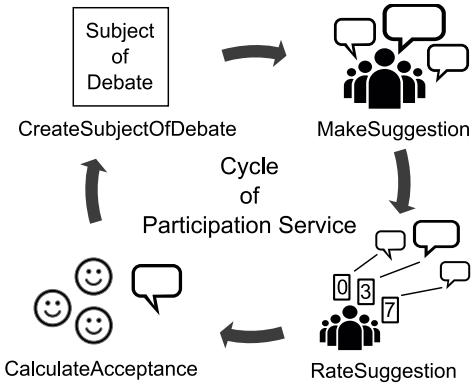


Figure 1. Systemic consenting process.

For illustration of our scenario-based requirements engineering technique, the simple goal "Rate a suggestion" of the Participation Service was chosen: A participant requests the website of the Participation Service and gets to see a login screen. After he logged in correctly, he gets a list of subjects of debate. He selects a subject of debate, which he is interested in. He sees a description of the subject and a list of suggestions sorted descending by acceptance. Once reading all suggestions, the participant rates each suggestion with refusing points from zero to ten to express his dislike against the suggestion. The Participation Service updates the acceptance of each suggestion and rearranges them.

IV. QUALITY-ORIENTED REQUIREMENTS ENGINEERING FOR AGILE DEVELOPMENT OF RESTFUL PARTICIPATION SERVICE

In this section, our requirements engineering methodology is introduced. This represents our proposed solution for gathering requirements that verifiably fulfill quality attributes introduced in ISO/IEC/IEEE 29148 [5]. This can be proven to the customer. First, the quality characteristics of the standards [4] and [5] are presented. Next, the stakeholders are identified followed by an elicitation of their goals. With the prioritization of the goals, they are selected for the iteration. Afterwards, the functional, non-functional requirements are discovered and documented according to the derived quality characteristics of [5] and the provided taxonomy by Glinz [9]. The entire requirements engineering methodology is shown in Fig. 2.

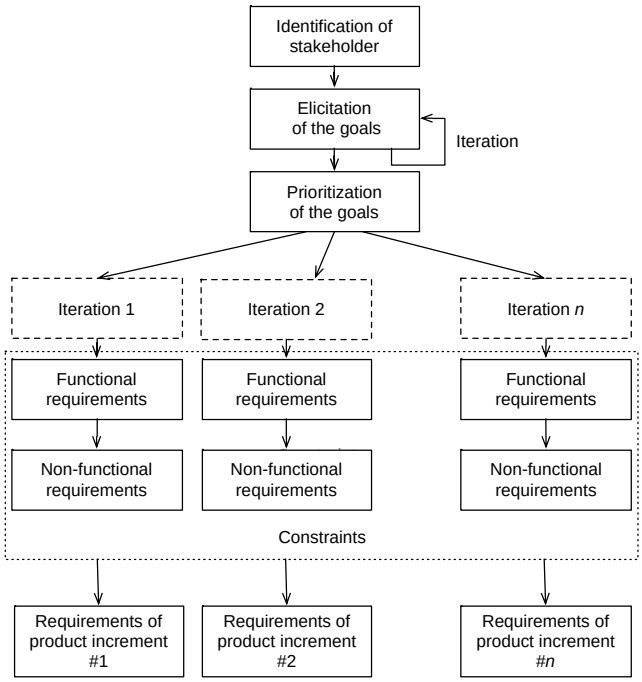


Figure 2. Requirements engineering methodology for agile development of RESTful Participation Service.

A. Quality Characteristics for Requirements

According to the IEEE [4], the requirements quality focuses on correctness, unambiguousness, completeness, consistence, prioritization, verifiability, modifiability, and traceability. [4] was replaced by the international standard ISO/IEC/IEEE 29148 [5], which introduces feasibility, necessity, free of implementation, and singularity as new characteristics for requirements while removing prioritization, correctness and modifiability. Furthermore, the new standard distinguishes between individual and a set of requirements. According to them, a set of requirements shall be complete, consistent, affordable, and bounded. These can be fulfilled by ensuring the individual ones. In the following, we consider the full set of quality characteristics for individual requirements of the current standard [5].

B. Identification of Stakeholders

In the elicitation phase, all stakeholders of the project have to be identified. A missing stakeholder can lead to incomplete requirements, which endanger the project success. For this purpose, we apply the approach by Sharp et al. [7]. Based on the four groups a) users, b) developers, c) legislators, and d) decision-makers, for the Participation Service, we could identify all stakeholders as listed in Table I and assign them to the corresponding scrum role.

The prioritization of the stakeholders with regard to their influence on the project was not necessary at this point. Due to the fact that the complexity of the project and the amount of involved stakeholders is not as high as in an industrial project.

 ISBN: 978-1-61208-367-4

TABLE I. STAKEHOLDERS OF THE PARTICPATION SERVICE

Group	Stakeholders
Users	Enrolled students and members of the KIT
Developers	Students at C&M and KIT as operator of the Participation Service
Legislators	State of Baden-Wuerttemberg and Federal Republic of Germany
Decision-Makers	C&M leader, C&M members and one expert of systemic consenting

C. Elicitation of Goals

After the identification of stakeholders, the elicitation of goals can be initiated. For this purpose, the interview and brainstorming technique was chosen and the questions introduced by Bruegge and Dutoit [12] were used for easier discovery of the goals according to the definition by [11]. Each goal corresponds exactly to one requirement in order to fulfill the singularity according to [5]. An excerpt of the determined goals is shown in Table II. Goal *G2* will be further refined in the upcoming sections.

TABLE II. EXCERPT OF GOALS OF THE PARTICIPATION SERVICE

ID	Goal	Stakeholder
G1	Logs in at the Participation Service	C&M member
G2	Rate a suggestion	C&M member
G3	Add a new proposal for solution	C&M member

In contrast to traditional software methodologies, such as the waterfall approach, in agile development more goals can be added in the course of the software project.

By investigating the quality characteristic of the current standard [5], we discovered that the meaning was changed compared to [4]. In [4], requirements were expected to be complete for the entire system. According to the current standard, a set of requirements contains everything to define a system or only a system element. This allows us, to use iterations in which system elements are described.

D. Prioritization of Goals

The next step is the prioritization of the goals with regard to their importance for the stakeholders. Due to the abstraction level of the goals and the statement by Wiegers [14], we applied a simple classification approach based on a three-level scale (essential, conditional, optional) according to [4]. In order to prevent ambiguousness, each stakeholder has agreed on the meaning of each level [14]. After rating of goals, a specific amount of highest ranked goals, which reflects the necessity [5], form the basis for the first iteration. The amount depends on the estimated velocity of the development team and expected effort for the implementation. In this context, the essential goals are those presented in Table II.

E. Functional Requirements

For each selected goal, a scenario will be authored or reused that describes the required interactions to reach the goal. Based on a scenario, further goals can be derived. The combination of a goal and the corresponding scenario is called requirement chunk as described in [11]. Fig. 3 illustrates this by showing a meta-model that defines the rules and the elements of a requirement chunk.

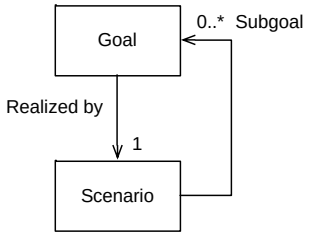


Figure 3. Meta-model of a requirement chunk.

This recursive process with objective of functional decomposition can be aligned with the process defined in the standard [5]. But, this recursive process can be repeated several times, which results in rising costs.

For that reason, we propose three conditions that serve as abort criteria for the process. If all of the following conditions apply, the process can be aborted:

1) no additional benefit in form of new derived goals
2) other scenarios will definitively not reuse atomic actions of the current scenario
3) the size of the scenario exceeds more than 20 atomic actions

According to Glinz [15], the decomposition in user functions and the ease of understanding assure the precondition of correct specification. Furthermore, the decomposition allows us to describe the capability and properties of a given requirement chunk in detail according to the stakeholder's need, which represents the completeness of individual requirements. In the following, authoring and reusing of scenarios will be presented.

E.1. Reusing Scenarios

In the best case, a requirement chunk still exists in the repository, which contains all analyzed goals and their scenarios. Therefore, redundant scenarios will be avoided, which ensures the consistence regarding to a set of requirements. As a result, we can compose different requirement chunks to support higher goals. For example, the goal *G1 "Logs in at the Participation Service"* represents a cross-sectional goal, which will be used by *G2* and *G3*. Furthermore, the usage of cross-references results in an increase of consistence [5]. This is because scenarios can be related to one another in a meaningful way, which allows detection of conflicts.

 ISBN: 978-1-61208-367-4

E.2. Authoring Scenarios

If no requirement chunk for the given goal can be found in the repository, a new scenario has to be authored while considering the quality characteristics by [5].

The unambiguousness cannot be fulfilled properly as we use the natural language with inherent equivocality for the description of the scenario [4]. So a trade-off between ease of understanding and formalism has to be made. For this, we used the provided meta-model of a scenario by Rolland et al. [11] to reduce equivocality. Moreover, we used the introduced structural constructs of Glinz [15] to further reduce the level of equivocality. To detect ambiguousness during description or validation of scenarios, Terzakis [16] offers a detailed checklist. Also, the current standard [5] provides some terms, such as superlatives or vague pronouns, which should be prevented to ensure bound and unambiguousness. For newly introduced terms and units of measure, we have created a separate document, which acts as a glossary.

According to [5], a scenario should be implementation free. This means that no architectural design decisions take place in this phase. This is the nature of a scenario as it describes what is needed in form of a concrete instance to achieve its intended goals. The nature of a scenario also allows us to derive acceptance criteria to verify the requirements in the form of test cases [15], which fulfills the verifiability [5].

The feasibility is another quality characteristic of the standard [5] with focuses on technical realization of the requirement. At this point, the scenario has to be investigated with regard to system constraints such as the existing environment (cf. Section G).

Title:	Rate a proposed suggestion	ID: G2	Priority: High
Source:	C&M member	Risk: Middle	Difficulty: Nominal
Rationale:	Integral ingredient of systemic finding	Version: 1.0	Type: Functional

Initial State:	User wants to rate a proposed solution
Final State:	User rated a proposed solution
Dependable goals:	No dependable goals

Nr.	Normal action flow	Ref.
1	User logs in at the Participation service	G1
	System verifies the credentials	
2	System redirects him to the secured area (Def. 1.1)	-
	User gets a list of available subjects of debate	
3	User selects a subject from the provided list	-
	System receives the selection and redirects him to the subject of debate	
4	User rates a proposed solution by selecting the refusing points	G5
	System calculates the acceptance of the suggested solution	
Nr.	Concurreny / Alternative action flow	
2'	*IF* the list of available subjects is empty *THEN* the system displays „There are currently no subjects of debate“ *TERMINATE*	

Figure 4. Style for representation of scenarios.

To ensure the traceability [5], each scenario must have a unique identifier. In the course of modification over time, the scenarios also need a version number representing the current state. Due to the fact of reusing scenarios, each scenario should also be aware of dependable requirement chunks to clarify, which requirement chunks will be affected by modifications of one scenario.

Based on these findings, the representation in [15], and the provided requirement attributes in [5], we created a style for representation of scenarios, which is illustrated in Fig. 4. Similar to the approach by Glinz [15], the representation can also be easily transformed into a state chart.

F. Non-Functional Requirements

After all goals have been analyzed, the resulting requirement chunks represent the functional aspects of the system. Each scenario can now be investigated with regard to non-functional aspects. For this purpose, we use quality attribute scenarios by Ozkaya et al. [17] and link these with the corresponding requirement chunk.

The stimulus represents the condition for the release of the event, while its source is the entity that triggers it. The response is the activity of the stimulus. The environment, such as normal operation of a service, stands for the constraint under which the stimulus occurred. The functional scenario represents the stimulated artifact. Finally, the response measure represents the measure for evaluating the response of the system.

To align this with the product strategy, the product quality characteristics [10] have to be ranked by their importance for the stakeholders. For example, the security is probably more important than the user experience for a product in the bank sector. This is why we used pairwise comparisons of the quality attributes according to the Analytical Hierarchy Process (AHP) by Saaty [18].

The results have shown that for the Participation Service security, functionality and usability are more important than the others. Based on this result, we could focus on the most important quality attributes. Nevertheless, we still have to keep the quality attributes with minor importance for the product strategy in mind. We can thus reduce the effort for eliciting the non-functional requirements since resources, such as time, often limit a project.

Type:	Usability	ID: N2	Priority: 0.18
Source:	C&M member, students	Risk: Low	Difficulty: Easy
Rationale:	Better user experience	Version: 1.0	Ref: G2
Quality attribute scenario	*Source of stimulus:*	User	
	Stimulus:	clicks on the button	
	Environment:	during normal operation,	
	Response:	the system gives a feedback	
	Response measure:	within a period of 200ms	

Figure 5. Style for representation of quality attribute scenarios.

 ISBN: 978-1-61208-367-4

Similar to the description of the functional scenarios (c.f. Section E), we have to respect the same conditions. This is why we do not describe this in detail at this point.

For the prioritization of non-functional requirements, we used the ranked result of the AHP. But, it is also possible to add another prioritization step, such as the ones mentioned in [14] or [17]. Fig. 5 shows one non-functional requirement of goal *G2*.

G. Constraints

According to Glinz [9], the constraints restrict the solution space for the functional and non-functional requirements. For example, a constraint can be company-based human interface guidelines, legal issues, or existing environments [9]. With regard to the Participation Service, we only had to investigate the constraints emerging from the existing environment: As described in the introduction, the Participation Service should be a part of the existing service-oriented KIT Smart Campus System based on REST. The usage of REST as an architectural style requires the consideration of six specific characteristics according to Fielding [1]: client-server, stateless, caching, uniform interface, layered architecture, and optionally code on demand. These constraints were written down in a separate constraints document similarly to the glossary so that we are able to reference this over the whole iteration cycle with regard to the feasibility [5].

V. Conclusion and Outlook

In this article, we introduced a requirements engineering methodology that is based on existing approaches and considers quality characteristics of the ISO/IEC/IEEE 29148 standard [5]. For that purpose, we analyzed the quality characteristics of [5] and enhanced existing methodologies for scenario-based requirements engineering.

We illustrated our approach by means of the Participation Service developed at the KIT. By applying our methodology on the Participation Service we could improve the quality of our requirements. For example, we detected some inconsistencies during the authoring of the scenarios and reduced the communication effort and the costs emerged from misunderstandings.

Compared to the previous IEEE recommendation [4], it is easier to meet the desired qualities of ISO/IEC/IEEE 29148 [5]. The reason for this is that the new standard does not give tough specifications for the satisfaction of the quality characteristics. Due to the fact that in a scenario-based approach we are using the natural language for describing requirements, we can only merely reduce the ambiguousness and not prevent it completely. However, this does not imply bad requirements but rather potential for improvements.

Our approach helps requirements engineers and business analysts with capturing and describing high-quality functional and non-functional requirements in a systematic manner. The quality characteristics are standardized [5] and represent recognized criteria for requirements. With our approach, requirements engineers and business analysts can capture new requirements that fulfill these criteria in a systematic manner or improve existing requirements.

In this article, we have focused on the requirements analysis phase of RESTful services. For the future, we plan to focus on the quality assurance of RESTful services during the design phase as part of an agile development process. After we have shown how to assure the quality of functional and non-functional requirements that constitute the basis for the development, the design phase has to consider quality as well. We will examine how to evaluate a service design as the one for the Participation Service regarding widespread quality characteristics and design patterns for RESTful services. For that purpose, we will enhance our existing work in the context of quality assurance of service-oriented architectures [19].

References

[1] R. Fielding, “Architectural styles and the design of network-based software architectures,” University of California, Irvine, 2000.

[2] Standish group, “Chaos report,” http://www.projectsmart.co.uk/docs/chaos-report.pdf, 1995, Accessed 2014-05-21.

[3] A. F. Hooks and K. A. Farry, “Customer centered products: creating successful products through smart requirements management,” American Management Association, 2000, ISBN 978-0814405680.

[4] IEEE, IEEE Std 830-1998 “Recommended practice for software requirements specifications,” 1998.

[5] ISO/IEC/IEEE, ISO/IEC/IEEE 29148:2011 “Systems and software engineering – life cycle processes – requirements engineering,” 2011.

[6] B. Nuseibeh and S. Easterbrook, “Requirements engineering: a roadmap,” The Future of Software Engineering, Special Volume published in conjunction with ICSE, 2000, pp. 35-46.

[7] H. Sharp, A. Finkelstein, and G. Galal, “Stakeholder identification in the requirements engineering process,” Database and Expert Systems Applications, 1999, pp. 387-391.

[8] F. Ackermann and C. Eden, “Strategic management of stakeholders: theory and practice,” Long Range Planning, Volume 44, No. 3, June 2011, pp. 179-196.

[9] M. Glinz, “On non-functional requirements,” 15th IEEE International Requirements Engineering Conference (RE 2007), 2007, pp. 21-26.

[10] ISO, ISO/IEC 25010:2011 “Systems and software engineering - systems and software quality requirements and evaluation (SQuaRE) - system and software quality models,” 2011.

[11] B. C. Rolland, C. Souveyet, and C. B. Achour, “Guiding goal modeling using scenarios,” IEEE Transactions on Software Engineering, Volume 24, No. 12, 1998, pp. 1055-1071.

[12] B. Bruegge and A. H. Dutoit, “Object-oriented software engineering: using uml, patterns and java,” Pearson Education, 2009, pp. 166-168.

[13] J. Karlsson and K. Ryan, “A cost-value approach for prioritizing requirements,” IEEE Software, Volume 14, No. 5, 1997, pp. 67-74.

[14] K. Wiegers, “First things first: prioritizing requirements,” Software Development, No. 9, Volume 7, Miller Freeman, Inc, September 1999, pp. 48-53.

[15] M. Glinz, “Improving the quality of requirements with scenarios,” Proceedings of the Second World Congress on Software Quality, Yokohama, 2000, pp. 55-60.

[16] J. Terzakis, “Tutorial writing higher quality software requirements,” ICCGI, http://www.iaria.org/conferences2010/filesICCGI10/ICCGI_Software_Requirements_Tutorial.pdf, 2010, Accessed 2014-07-16.

[17] I. Ozkaya, L. Bass, R. L. Nord, and R. S. Sangwan, “Making practical use of quality attribute information,” IEEE Software, April 2008, pp. 25-33.

[18] T. L. Saaty, “How to make a decision: the analytic hierarchy process,” Informs, Volume 24, No. 6, 1994, pp. 19-43.

[19] M. Gebhart, “Measuring design quality of service-oriented architectures based on web services,” Eighth International Conference on Software Engineering Advances (ICSEA 2013), Venice, Italy, October 2013, pp. 504-509.

 ISBN: 978-1-61208-367-4

Architecture Coverage: Validating Optimum Set of Viewpoints

Sunia Naeem, Salma Imtiaz
Department of Computer Science and Software Engineering,
International Islamic University,
Islamabad, Pakistan
sunia.naeem@yahoo.com, salma.imtiaz@iiu.edu.pk

Abstract— There are various software architecture viewpoint models but none of them provides optimum coverage of software architecture domain. Software architecture coverage is the coverage of concepts that are required to effectively design and analyze software architecture. An optimum set of viewpoints can be selected from different software architecture viewpoint models that provide maximum coverage of software architecture domain than an individual architecture model. In this paper, an optimum set of viewpoints is selected by comparing five commonly used software architecture viewpoint models namely 4+1 RM-ODP, SEI, Siemens and Rational ADS via a common comparison framework. These architecture models are compared on the evaluation criteria, i.e., viewpoints, stakeholders and quality attributes. This evaluation criterion is based on IEEE Standard 1471 Recommended Practice for Software Architecture Description. The resulting optimum set is validated in industry via multiple case studies, and the results show that the optimum set of viewpoints provide greater coverage than any of the viewpoint alone.

Keywords-architecture coverage; optimum; viewpoints; stakeholders; quality attributes.

I. INTRODUCTION

Software architecture is system's high level structure and describes that system as computational components and interactions between them.

The need for documenting software architecture is its ability to communicate between stakeholders, to provide re-usable abstractions of software systems and to capture early design decisions [1]. The commonly used approach to model a complex architecture was to make use of a heavily overloaded, single model that does not adequately represent the system and difficult to understand and manage [2]. Some of the disadvantages of using this approach are unreliable notations, over emphasis of one aspect, mixing of architectural styles and overlooking of individual stakeholder concerns [1].

A great amount of work has been done to partition the architecture of the system into multiple views, where each view highlights a different perspective. This approach helps in comprehension and understandability from stakeholders' point of view. Architects also come to an understanding that to develop successful software architecture we should draw many different system structures simultaneously to handle the multi-faceted nature of architecture. It seems that software research community also have decided that the only way to design architecture is by representing system using several related models (or views) [3].

Viewpoints are used to choose which view to produce for a particular system, and what information to represent in that view. Views and viewpoints usage has various benefits, such as management of complexity, separation of concerns and improved communication with stakeholders. Viewpoint model [3] means a framework that describes the significant concerns that need to be taken care of while designing software architecture. Generally, software architecture models contain several viewpoints which define the models and concepts which can be used while dealing with the specific concern.

A research work by Nicholas May [1] surveys the different viewpoint models and highlights that existing viewpoint models need to be tailored because they do not address every concern of software architecture domain. The key purpose of this research work was to understand different software architecture models, their coverage of software architecture domain and their comparative strengths. The view point models are compared with respect to IEEE 1471-2000 Standard called the IEEE Recommended Practice for Software Architecture Description.

The author also proposed a classification of viewpoints within a common framework that allowed combining views from different viewpoint models and determining an optimum set of views with the purpose of providing maximum coverage to represent the architecture. Different vocabularies of models can be compared by common reference vocabulary. Optimum set, has the maximum coverage as compared to any individual viewpoint model.

Viewpoint models selected in this survey[1] are Kruchten's "4+1" View Model [20], Siemens Four View model [23], Software Engineering Institute (SEI) set of views [21], Rational Architecture Description Specification (ADS) [25] and ISO Reference Model of Open Distributed Processing (RM-ODP) [22]. All these five models describe software architecture from multiple perspectives. Each one of them identifies separation of concerns and specifies stakeholders. Also, these models focus on describing software architecture structures instead of describing particular notations for each of these structures.

 ISBN: 978-1-61208-367-4

Our research work focus on the extension of comparison criteria described in [1]. Section II presents the related work and compares it on a common criteria. The comparison criteria is presented in Section III where the mapping of stakeholder, viewpoints and quality attributes is presented in Section IV. The evaluation on the chosen criteria is done in Section V, and the proposed optimum set of viewpoints is presented in Section VI. The evaluation in this extended coverage criteria done in Section VII and validated in Section VIII. Conclusion and future work are discussed in Section IX.

II. RELATED WORK

This section, presents the comparison of different viewpoint models based on the ***Focus of Research, Criteria used for Comparison and Limitation of the Research Work.***

TABLE I OVERVIEW OF RELATED WORK

Paper Id	Description	Models reviewed/compared	Criteria for comparison	limitations
[3]	In this paper, the author surveyed some architecture Models and conduct a case study on the usage of software architecture documentation practices in the Telecommunications industry.	RM-ODP, US Department of Defense frameworks TAFIM, C4ISR, 4+1 view, Zachman framework	No specific criteria	Models are reviewed from literature and their details, benefits and deficiencies are based on literature review.
[4]	In this paper, viewpoint sets are applied to development of information systems and evaluated so weaknesses and strengths of every set of viewpoints is described and few general observations about their definition and use are presented.	4+1, RM-ODP, Siemens, Garland and Anthony	Industrial experience	Comparison is based only on observations of author. No common reference vocabulary is used for comparison.
[5]	This study provides analysis and comparison of six architecture frameworks categorized by major elements such as their inputs, outcomes and goals. It provides classification of architecture frameworks into Software Architecture Frameworks and Enterprise Architecture Frameworks and identifies some of their deficiencies.	Zachman Framework, 4+1, Federal Enterprise Architecture Framework (FEAF), RMODP, Department of Defense Architecture Framework (DoDAF), The Open Group Architecture Framework (TOGAF)	goals, inputs and outcomes	More focus is on classification of frameworks not on frameworks' deficiencies.
[26]	In this paper, the author provides overview of two classes of architecture frameworks Software Architecture Frameworks and Enterprise Architecture Frameworks and find some dimensions which can be helpful to understand architecture documents	Zachman Framework, The Information Framework (IFW), Integrated Architecture Framework (IAF), The Open Group Architecture Framework (TOGAF), Methodology forArchitecture Description (MAD), 4+1, Siemens	No specific criteria	More focus of comparison is on the difference between two classes of architecture framework.
[27]	In this paper, the author surveyed few architecture frameworks and compared them on the basis of methodologies and techniques they use and suggested that more architecture styles can be added to yield new architecture framework which focus on quality	Zachman Framework, 4+1, Federal Enterprise Architecture Framework (FEAF), RM-ODP, Department of Defense Architecture Framework (DoDAF), The Open Group Architecture Framework (TOGAF)	methodologies and techniques used in the framework	Focus of this comparison is to state only general advantages and disadvantages of architecture frameworks
[6]	This paper compares SEI with IEEE 1471 and show compliance of SEI with IEEE 1471.	SEI, IEEE 1471	Requirements imposed by IEEE 1471	Only compliance of one viewpoint model is considered .Compliance of other viewpoint models are missing.

III. COMPARISON FRAMEWORK ELEMENTS

To compare the software architecture viewpoint models, a common comparison framework is required. IEEE 1471-2000 Standard, called the IEEE Recommended Practice for Software Architecture Description [19] has been selected for evaluation, which consists of viewpoints, stakeholders and quality attributes, and their relationships for documenting the software architecture. IEEE 1471 considers stakeholders and their respective architectural concerns as essential elements in an architectural description. Architectural concern is a matter of importance to one or more stakeholders relating to the architecture. Another major element of ANSI/IEEE 1471 is that every architecture view in an architecture description is defined relative to an architectural viewpoint as we know architecture description is planned into multiple views and each one of them denotes the system architecture with reference to a set of related architectural concerns. So an architecture viewpoint captures the rules for analyzing and constructing a particular view and acts as a view template so it can be reused across many architectural descriptions.

A. Software Architectural Viewpoints

Viewpoints reason about quality attributes so architecture description should provide enough details or information necessary to analyze quality attributes. We have added conceptual viewpoint, in the list of viewpoints stated by Nicholas. Conceptual viewpoint [8] describes the system in form of system's major design elements and relationship between them. Conceptual viewpoint is very important because it is strongly linked with the problem domain and acts as an important means of communication when the architect interacts with domain expert. It helps in clearly defining modules in module view and impact of changes in requirements can be minimized. Viewpoints are not system specific so they are pre-defined and reusable.

B. Software Architectural Stakeholders

Stakeholder of software architecture is someone who has a vested interest in it, who implicitly or explicitly motivates the whole shape and direction of the architecture [16].

Stakeholders are consumers of software architecture description and architecture description serves as a means of communicating design decisions between stakeholders. Architecture should be communicated in a way that stakeholders use it properly for their respective use [24]. There is variety of stakeholders and their use with respect to architectural documentation varies. Nicholas`s list [1] of stakeholders can be extended to incorporate all stakeholders meant for required comparison. Our analysis will be based on the stakeholders who are consumers of software architecture's documentation. These stakeholders will make the analysis of viewpoints possible as they provide coverage of stakeholders that different viewpoints address e.g., product managers, business analysts and marketers.

C. Software Quality Attributes

Software architecture description should address stakeholder's concerns otherwise it is considered incomplete [16].

Concerns [18] are normally driven by the need for the system to exhibit a certain quality attributes rather than to provide a particular function. There is inherent need to consider quality attributes in each architecture view. Quality attributes are considered as concerns. Quality attributes can be classified into three types: Run-time, development-time and business. Nicholas`s list of concerns [1] does not include important concerns such as business quality attributes which repeatedly form a system`s architecture. Table 2 shows elements of our comparison framework comprising viewpoints, stakeholders and quality attributes.

TABLE II ELEMENTS OF COMPARISON FRAMEWORK

Viewpoints[12][8]	**Stakeholders[12][24][10][11]**	**Quality Attributes[17][12]**
Conceptual Decomposition Uses Layered Class/Generalization Process Concurrency Shared Data Client-Server Deployment Implementation Work Assignment	Architects Requirements Engineers Sub-System Architects and Designers Implementers Testers Integrators Maintainers External System Architects and Designers Managers Product Line Managers Quality Assurance Team Users Customers Project Manager Production Engineers Suppliers System Administrators Business Analysts Product Managers Marketers Support Staff	**System Run-Time** Functionality Performance Capacity/Space Availability Reliability Security Safety Usability Supportability configurability Scalability Interoperability **System Development-Time** Modifiability Reusability Testability Portability Evolvability Localizability Integrability **Business** Time to market Cost and benefit Projected lifetime of the system Targeted market Rollout schedule Integration with legacy systems

IV. MAPPING BETWEEN STAKEHOLDERS, VIEWPOINTS AND QUALITY ATTRIBUTES

Thus, evaluation done by Nicholas [1] can be extended on all three attributes which are viewpoints, stakeholders an quality attributes. In case of stakeholders and quality attributes, only those are covered that are explicitly stated by viewpoint models. We identify implicit quality attributes and stakeholders by investigating the relationship between stakeholders,viewpoints and quality attributes. Implicit stakeholders will be satisfied if all their concerns are

 ISBN: 978-1-61208-367-4

addressed by viewpoints and similarly different viewpoints address different quality attributes.

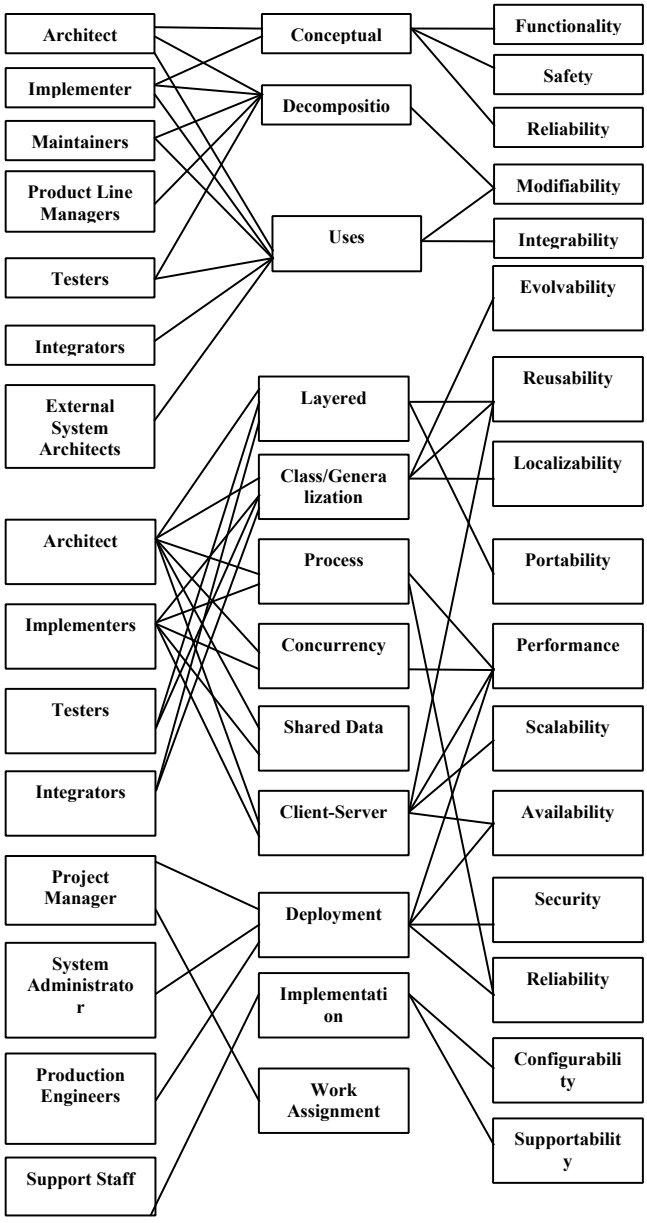


Figure 1. Mapping between Stakeholders, Viewpoints and Quality Attributes

V. MODELS EVALUATION AND COMPARISON FRAMEWORK COVERAGE

Tables 3, 4 and 5 show the coverage of viewpoints, stakeholders and quality attributes by the five software architecture viewpoint models. The coverage is found individually for each of the elements of comparison framework's concepts of stakeholders, quality attributes and viewpoints. Each viewpoint model provides different coverage of comparison framework elements. In case of viewpoints and quality attributes, SEI provides greatest coverage. As far as stakeholders are concerned, SEI and Rational ADS provide good coverage of stakeholders.

TABLE III MODELS COVERAGE OF VIEWPOINTS

Viewpoints	"4+1"	SEI	RM-ODP	Siemens	Rational ADS
Conceptual	Y	N	Y	Y	Y
Decomposition	Y	Y	Y	Y	Y
Uses	N	Y	N	N	N
Layered	Y	Y	Y	Y	Y
Class/Generalization	Y	Y	Y	N	N
Process	Y	Y	N	Y	Y
Concurrency	Y	Y	Y	Y	Y
Shared Data	N	Y	Y	N	N
Client-Server	Y	Y	Y	Y	N
Deployment	Y	Y	Y	N	Y
Implementation	N	Y	N	Y	Y
Work Assignment	N	Y	N	N	N

TABLE IV MODELS COVERAGE OF STAKEHOLDERS

Stakeholders	"4+1"	SEI	RM-ODP	Siemens	Rational ADS
Architects	Y	Y	N	Y	Y
Requirements Engineers	Y	N	N	Y	Y
Sub-System Architects and Designers	Y	Y	N	Y	Y
Implementers	Y	Y	Y	N	Y
Testers	Y	Y	Y	N	Y
Integrators	Y	Y	Y	Y	Y
Maintainers	N	Y	Y	N	N
External System Architects and Designers	N	Y	N	N	N
Managers	Y	Y	Y	Y	Y
Product Line Managers	Y	Y	N	N	N
Quality Assurance Team	N	N	N	N	Y
Users	Y	N	Y	N	Y
Customers	Y	N	Y	N	Y
Project Manager	N	Y	N	N	N
Production Engineers	Y	Y	Y	N	Y
Suppliers	N	Y	N	N	N
System Administrators	Y	Y	Y	N	N
Business Analyst	N	N	Y	N	Y
Product Manager	N	Y	Y	N	N
Marketer	Y	N	Y	Y	Y
Support Staff	N	Y	N	Y	Y

TABLE V MODELS COVERAGE OF QUALITY ATTRIBUTES

Quality Attributes	"4+1"	SEI	RM-ODP	Siemens	Rational ADS
Functionality	Y	N	Y	Y	Y
Performance	Y	Y	N	Y	N
Capacity/Space	N	Y	Y	N	Y

 ISBN: 978-1-61208-367-4

Availability	Y	Y	N	Y	Y
Reliability	Y	Y	N	Y	Y
Security	N	Y	Y	N	N
Safety	N	N	N	Y	N
Usability	N	N	N	N	Y
Supportability	N	Y	N	Y	Y
Configurability	N	Y	N	Y	Y
Scalability	Y	Y	N	N	Y
Modifiability	N	Y	Y	N	N
Reusability	Y	Y	N	N	Y
Testability	N	N	Y	N	Y
Portability	Y	Y	Y	Y	Y
Evolvability	Y	Y	Y	N	N
Localizability	Y	Y	N	N	N
Integrability	N	Y	N	N	N
Interoperability	Y	Y	N	Y	Y
Time to market	Y	Y	Y	Y	Y
Cost and benefit	Y	N	Y	Y	Y
Projected lifetime of the system	N	Y	Y	N	N
Targeted market	Y	N	Y	Y	Y
Rollout schedule	N	Y	Y	N	N
Integration with legacy systems	Y	N	N	N	Y

Y: provides Coverage
N: Does not provide coverage

VI. Optimum Set Of Viewpoints

When combining views from different viewpoint models, the biggest obstacle is dependency between views of viewpoint models. In case of" 4+1" model the views are dependent on each other, i.e., being an iterative method there is strong data flow between views. The views of the SEI and the RM-ODP model are comparatively independent. The views of Siemens model are less tightly coupled. In Rational ADS, context of lower views are provided by higher views so there is strong dependency between views.

Therefore, when combining views from different viewpoint models, we see that SEI model provides good coverage of viewpoints, stakeholders and quality attributes and also its views are independent so its three views that are module, component and connector and allocation are considered for merging. The missing stakeholders such as users, customers and business analysts which are not addressed by SEI can be incorporated by including Use Case View from Rational ADS. There is a dependency between Rational ADS views as Use Case being the highest view is not dependent on any other view. Use Case View also covers the usability concern which is not covered by SEI model. Siemens's Conceptual view is also included in optimum set as SEI model does not cover the conceptual structure and its related concern, which is functionality. Conceptual viewpoint [8] describes the system in form of system's major design elements and relationship between them. This viewpoint is very important because it is strongly linked with the problem domain.

Rational ADS Test View is added in optimum set of views to address testability Rational ADS's Test view addresses testability by enabling one to perform test realization, preparing test cases and then forming whole test procedure also satisfying the Quality Assurance Team. As we know that in Rational ADS that context of lower views are provided by higher views so we investigated and found that SEI Allocation view type overlaps well with Rational ADS Realization viewpoint which contains Implementation and Deployment View. So, context of Test View can be provided by Allocation View type of SEI model. RM-ODP views are not considered for merging because RM-ODP uses language for architecture description and not a notation so it supports communication between different systems developers and not among other stakeholders of the same system. Figure 2 shows optimum set of views from different viewpoint models.

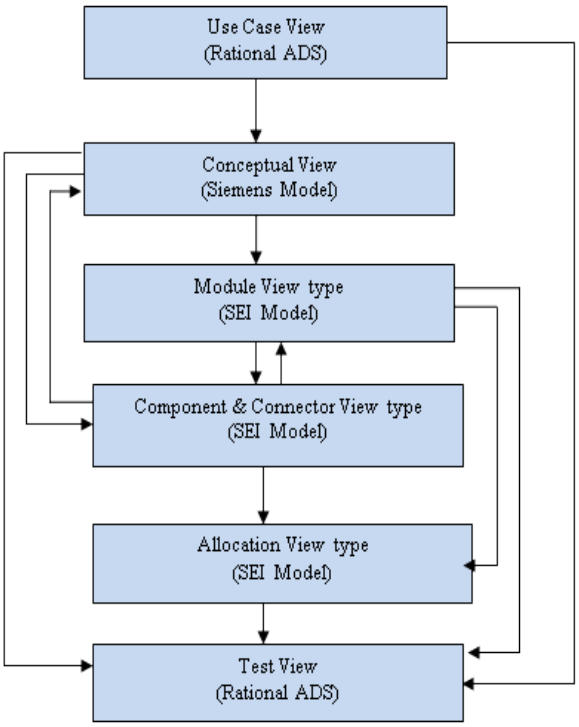


Figure 2. Optimum Set of Viewpoints

VII. Coverage of Optimum Set of Viewpoints

Tables 6 and 7 show stakeholders and quality attributes addressed by Optimum Set of Viewpoints.

 ISBN: 978-1-61208-367-4

TABLE VI STAKEHOLDERS ADDRESSED BY OPTIMUM SET OF VIEWPOINTS

Views	Stakeholders Addressed
Use Case	Users, Customers, Business Analysts
Conceptual	Architect, Implementers
Decomposition	Implementers, Maintainers, Product Line Managers, Architect, Testers
Uses	Implementers, Maintainers, Architect, Testers, Integrators, External System Architects and Designers
Generalization	Implementers, Architect, Testers, Integrators
Layered	Implementers, Architect, Testers, Integrators
Component And Connector	Implementers, Architect
Deployment	Project Manager, Testers, Integrators, Architect, System Administrator, Production Engineers
Implementation	Support Staff
Work Assignment	Project Manager
Test	Testers, Quality Assurance Team

TABLE VII QUALITY ATTRIBUTES ADDRESSED BY OPTIMUM SET OF VIEWPOINTS

Views	Quality Attributes Addressed
Use Case	Usability
Conceptual	Functionality, Safety, Reliability
Decomposition	Modifiability
Uses	Modifiability, Integrability
Generalization	Evolvability, Reusability, Localizability
Layered	Portability, Modifiability, Reusability
Pipe-and-Filter	Performance
Shared-Data	Security
Client-Server	Performance, Scalability, Availability, Reusability
Peer-to-Peer	High Availability, High Scalability
Communicating Processes	Performance, Reliability
Deployment	Performance, Reliability, Availability, Security
Implementation	Configurability, Supportability
Test	Testability

VIII. VALIDATION OF OPTIMUM SET OF VIEWPOINTS

A. Research Design

In order to validate optimum set of viewpoints, we conducted multiple-case study[7] of three software intensive projects of medium to large complexity whose architectures were built using our proposed optimum set of viewpoints either by software architect or personnel who have sound knowledge of developing software architecture by using software architecture viewpoint models. We have chosen a multiple-case study approach as multiple sources of evidence allow a better validity for the findings and used purposeful sampling. We looked for projects of those software development companies that had experience in using software architecture viewpoint models and also have experienced personnel who have sound knowledge of applying views for developing architecture of applications.

B. Data Sources

We collected data using semi-structured scripted interviews so the questions were prepared in advance and pre-defined questionnaire were used and filled in print. We could not manage to conduct face to face interviews or interview via Skype Out calls because of nature and secrecy of projects and work load.

C. Data Analysis

The purpose of filling the questionnaire was to find out optimum set of views coverage of software architecture concepts (i.e., viewpoints, stakeholders and quality attributes) that are required to efficiently design and analyze software architecture after applying it on the case projects and discuss its coverage as compared to the software architecture viewpoint model which they usually use to develop architecture of their applications. To analyze data, frequency distributions related to coverage of viewpoints, stakeholders and quality attributes by our research outcome i.e., optimum set of viewpoints in all three cases are developed separately in the form of graphs in section E .

D. Overview of Case Studies

1) Project A

Project A is software project developed by a software house (CMMI Level 3) that specializes in developing Financial, Business Management and E-government applications and project A is E-government in nature. Project A's architecture is built using optimum set of viewpoints by their software architect who has eight years experience in developing architecture of applications and Software Architect has used all views of optimum set to develop application's architecture due to project's complexity.

After analyzing data of questionnaire we found out that according to architect's views and analysis of questionnaire optimum set of viewpoints provide more coverage with respect to viewpoints and stakeholders' concerns as compared to the viewpoint model (i.e., Rational ADS with customization) which they usually follow for developing architecture because it ignores the internal structures of the application and hence the performance and reliability behaviors are not explicitly and individually captured, so these types of problems are sufficiently covered by optimum set of views. In case of quality attributes optimum set of views provides all applicable attributes. Suggestion given by Architect is that optimum set should define how things in one view are connected and complimented in the next view such as how uses cases are linked to class and sequence diagrams and how they are connected to test cases so an overall detailed inter connectivity needs to be defined.

2) Project B

Project B is software project developed by a software house (CMMI Level 2) that specializes in Data Management (Data warehouse, Business Intelligence, Data Mining,

 ISBN: 978-1-61208-367-4

Document Management Application Dev., Document Management Services) in Telecom and Banking Domains. They did not give much detail of project. Project B's architecture is built using optimum set of viewpoints by their Project Manager who has five years plus experience in developing architecture of applications and after that our questionnaire is filled by him in order to find coverage of optimum set of viewpoints. Project Manager has used all views of optimum set to develop application's architecture due to project's complexity.

After Analyzing data of questionnaire we found out that according to architect's view and analysis of questionnaire optimum set of viewpoints provide more coverage of business needs and maximum completeness of software architecture aspects i.e., viewpoints, stakeholders and quality attributes by customizing already available software architecture solutions. Being the SEI / CMMI certified firm they usually follow SEI's views with customization to work for implementation of data warehouse and business intelligence projects.

3) Project C

Project C is software project developed by a software house that specializes in managing the entire office automation system and providing IT support to defense organizations and project C is web based document management and filing system. Project C's architecture is built using optimum set of viewpoints by their project manager who has four years experience in developing architecture of applications. Software Architect has used all views of optimum set except Component & Connector View type to develop application's architecture.

After Analyzing data of questionnaire we found out that according to architect's views and analysis of questionnaire optimum set of viewpoints provide more coverage with respect to viewpoints and quality attributes as compared to the software architecture processes or models (i.e., RUP and Rational ADS with customization) which they usually follow for developing architecture. In case of quality attributes optimum set of views provides high availability as compared to approach followed by them. Suggestion given by project manager is use case viewpoint should be added in list of viewpoints.

E. Case Studies Results

1) Coverage of Viewpoints

Figure 3 shows coverage of software architecture viewpoints by optimum set of viewpoints after applying it on case projects. Out of 12 viewpoints optimum set of viewpoints provides 100% coverage, i.e., 12 viewpoints in first case study, 92% coverage, i.e., 11 viewpoints in second case study and 83% coverage, i.e., 10 viewpoints in third case study.

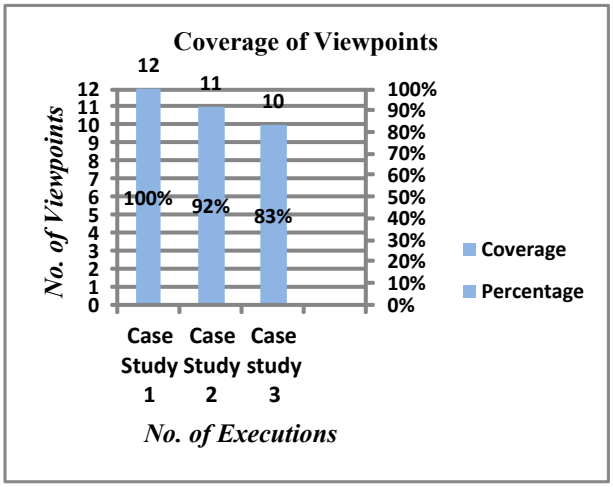


Figure 3. Coverage of Viewpoints by Optimum set of Viewpoints

From analysis of questionnaire results it is shown that viewpoints such as shared data, uses, generalization, implementation and work assignment which are not covered by most of models, are covered in detail by optimum set of viewpoints.

2) Coverage of Stakeholders

Figure 4 shows coverage of software architecture stakeholders by optimum set of viewpoints after applying it on case projects. Out of 21 stakeholders optimum set of viewpoints provides 100% coverage, i.e., 21 stakeholders in first case study, 100% coverage, i.e., 21 stakeholders in second case study and 76% coverage, i.e., 16 stakeholders in third case study.

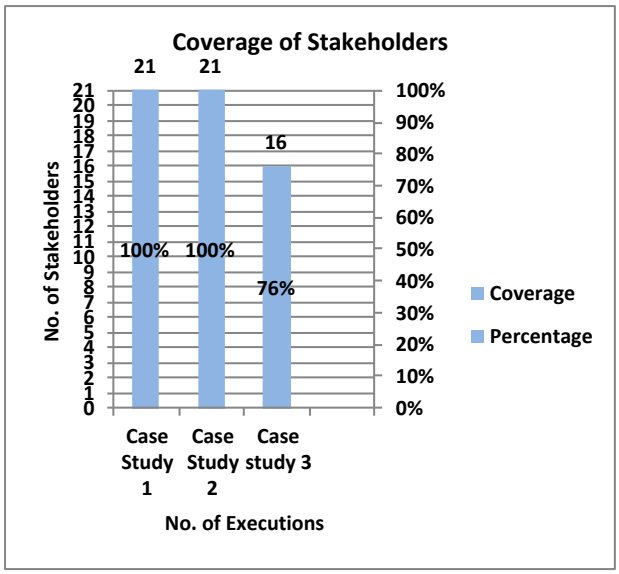


Figure 4. Coverage of Stakeholders by Optimum set of Viewpoints

From analysis of questionnaire results it is shown that stakeholders such as Designers, External System Architects Quality Assurance Team, Product line Managers, Suppliers, Support Staff and Project Managers which are not covered by most of models are covered in detail by optimum set of viewpoints.

3) Coverage of Quality Attributes

Figure 5 shows coverage of software architecture quality attributes by optimum set of viewpoints after applying it on case projects. Out of 25 quality attributes optimum set of viewpoints provides 100% coverage i.e., 25 quality attributes in first case study, 80% coverage i.e., 20 quality attributes in second case study and 96% coverage i.e., 24 quality attributes in third case study.

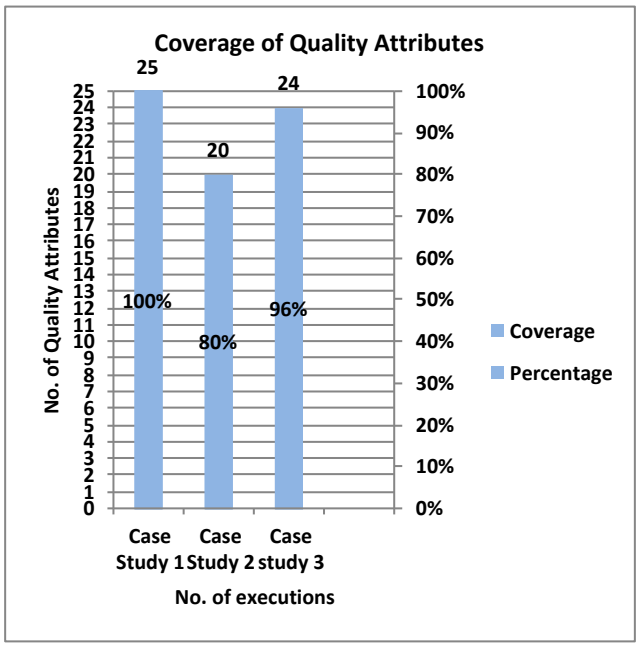


Figure 5. Coverage of Quality Attributes by Optimum set of Viewpoints

From analysis of questionnaire results it is shown that quality attributes such as security, modifiability, integrability, safety, supportability, projected lifetime of the system and testability which are not covered by most of models are covered in detail by optimum set of viewpoints.

F. Discussion

Mean coverage of concepts that are needed to efficiently design and analyze software architecture i.e., viewpoints, stakeholders and quality attributes is calculated for optimum set of viewpoints and compared to coverage of viewpoints by five software architecture viewpoint models and it is shown that optimum set of viewpoints provide more coverage of concepts than surveyed individual models.

Figure 6 shows comparison between optimum set of viewpoints and surveyed individual models with respect to coverage of viewpoints. Optimum set of viewpoints provide more coverage as compared to individual models. SEI coverage and optimum set of viewpoints coverage is same in case of viewpoints because our comparison framework is based on IEEE 1471 Standard i.e., Recommended Practice for Architectural Description of Software-Intensive Systems and SEI model provides template for more than one representation to describe contents of view in order to conform with the IEEE 1471 and can cover all details.

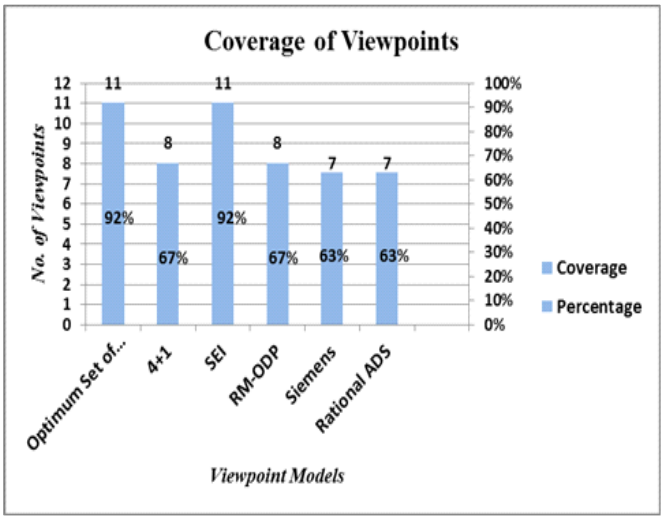


Figure 6. Comparison of Coverage of Viewpoints by Optimum set of Viewpoints with individual viewpoint models' coverage

Figure 7 shows comparison between optimum set of viewpoints and surveyed individual models with respect to coverage of stakeholders. Optimum set of viewpoints provide more coverage as compared to individual models.

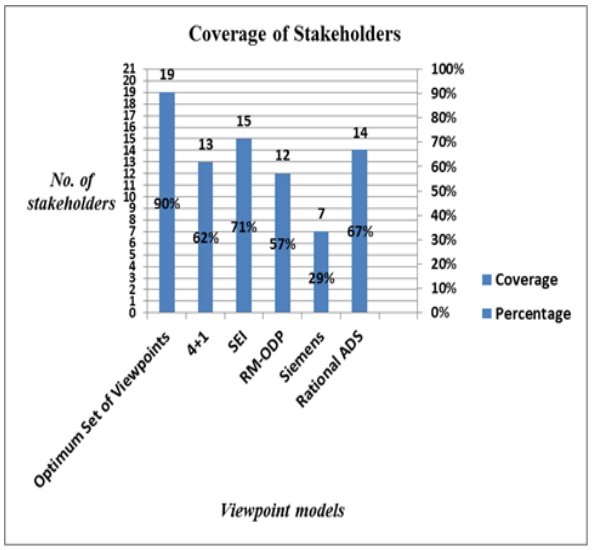


Figure 7. Comparison of Coverage of Stakeholders by Optimum set of Viewpoints with individual viewpoint models' coverage

 ISBN: 978-1-61208-367-4

Figure 8 shows comparison between optimum set of viewpoints and surveyed individual models with respect to coverage of quality attributes. Optimum set of viewpoints provide more coverage as compared to individual models.

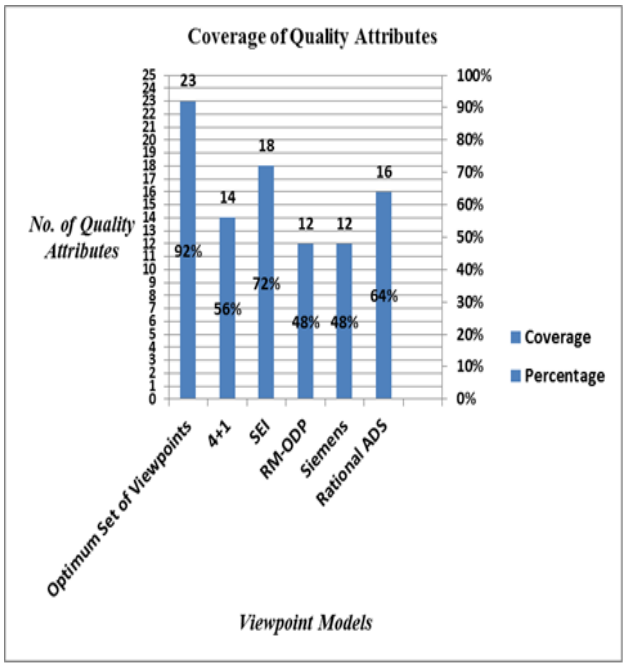


Figure 8. Comparison of Coverage of Quality Attributes by Optimum set of Viewpoints with individual viewpoint models' coverage

Also from case studies, it is concluded that Optimum set of views provide more coverage with respect to viewpoints, stakeholders and quality attributes of software architecture domain, than what can be achieved via individual architecture model alone.

G. Limitations

Due to resource limitations and confidentiality issues, we were not able to triangulate our findings by software architectural documentation analysis and face to face interviews which can provide in depth analysis. Furthermore, close ended questions in questionnaire has Yes\No\Partial\Not Applicable options, so while analyzing questionnaire results we assign same scale to partial option as Yes option regarding coverage of Software architecture concepts because we have to compare coverage of optimum set of viewpoints with coverage of surveyed viewpoint models whose coverage were determined by review of literature not by software architectural documentation analysis and from review of literature partial coverage cannot be find out.

IX. Conclusion And Future Work

A. Conclusion

There are a number of viewpoint models that create architecture document by means of the separation of the concerns. Each one of them describes viewpoints set and recognizes the concerns that each of them address. But none of them provides complete coverage of software architecture domain. So, a set of optimum viewpoints is selected from different software architecture viewpoint models after comparing them on a common comparison framework that allows combining views from different viewpoint models.

Then we present a Multiple-case study on the application of optimum set of viewpoints to three software development projects. From the results of case studies it is concluded that Optimum set of views provide more coverage with respect to viewpoints, stakeholders and quality attributes of software architecture domain, than what can be achieved via individual architecture model alone.

B. Future Work

In the future, this work can be augmented by additional case projects and data can be collected and analyzed from several sources i.e., architectural documentation and face to face interviews to get a more complete understanding of coverage of software architecture concepts.

Furthermore, by modeling system from architectural documentation with five surveyed models we can get a clearer picture of their coverage of software architecture concepts and also their partial coverage of concepts can be found, which cannot be found via literature.

References

[1] N. May, "A survey of software architecture viewpoint models", Proc. of the Sixth Australasian Workshop on Software and System Architectures, March 2005, pp. 13-24.

[2] N. Rozanski and E. Woods, "Applying Viewpoints and Views to Software Architecture", 2005.

[3] K. Smolander," What is included in software architecture? A case study in three software organizations", Proc. Of the Ninth Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, April 2002, pp. 131–138.

[4] E. Woods, "Experiences using viewpoints for information systems architecture: an industrial experience report", Proc. of Software Architecture First European Workshop EWSA, 2004, pp. 182-193.

[5] A. Tang, J. Han and P. Chen, "A Comparative Analysis of Architecture Frameworks", Proc. of the 11th Asia-Pacific Software Engineering Conference, 2004, pp. 640-647.

 ISBN: 978-1-61208-367-4

[6] P. Clements, ”Comparing the SEI's Views and Beyond Approach for Documenting Software Architectures with ANSI-IEEE 1471-2000”, Technical Note 017, Software Engineering Institute, Carnegie Mellon University, 2005.

[7] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering." Empirical software engineering, vol. 14.2, 2009, pp. 131-164.

[8] L. Bass and R. Kazman, “Architecture-Based Development”, Technical Report 007, Software Engineering Institute, Carnegie Mellon University, 1999.

[9] M. Shaw and D. Garlan, “Software Architecture: Perspectives on an Emerging Discipline”, Prentice-Hall, 1996.

[10] K. Smolander and T. Päivärinta , "Describing and Communicating Software Architecture in Practice: Observations on Stakeholders and Rationale", Proc. of the Fourteenth International Conference on Advanced Information Systems Engineering, 2002, pp. 117-133.

[11] Z. Baida, “Stakeholders and Their Concerns In Software Architectures”, October 2001.

[12] L. Bass, P. Clements and R. Kazman, “Software Architecture in Practice”, 2nd edition, Addison Wesley, 2003.

[13] S. Fricker, T. Gorschek and P. Myllyperkiö, “Handshaking Between Software Projects and Stakeholders Using Implementation Proposals”, Proc. of the 13th International Working Conference on Requirements Engineering: Foundation for Software Quality, Springer, 2007, pp. 144 – 159.

[14] W.H. Huen, “Systems Engineering of Complex Software Systems”, Proc. of 37th ASEE/IEEE Frontiers in Education Conference, October 2007.

[15] J Asundi, R. Kazman and M. Klein, “An Architectural Approach to Software Cost Modeling”, SEI Interactive, March 2000.

[16] N. Rozanski and E. Woods, “Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives”, Addison-Wesley, 2005.

[17] Malan, R. and D. Bredemeyer, “Defining non-functional requirements”, Bredemeyer Consulting white paper, 2001.

[18] F. Losavio, L. Chirinos, N. Lévyand and A. Ramdane-Cherif, “Quality Characteristics for Software Architecture”, Journal of Object Technology, Vol. 2, No. 2, 2003, pp. 133-150.

[19] IEEE. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. Institute of Electrical and Electronics Engineers, Sept. 2000.IEEE Std 1471-2000.

[20] P. Kruchten. “Architectural Blueprints - The “4+1” View Model of Software Architecture”, IEEE Software, vol. 12, Issue no. 6,1995,pp. 42–50.

[21] P. Clements et al. “A practical method for documenting software architectures”, Proc. Of the 25th ICSE, 2003.

[22] ISO. Reference Model of Open Distributed Processing (RMODP). International Organization for Standardization, Technical Report 10746, 1994.

[23] D. Soni, R.L. Nerd, and C. Hofmeister, “Software architecture in industrial Applications”, Proc. of International Conference on Software Engineering,1995, pp. 196–207.

[24] P. Clements et al., “Documenting Software Architecture: Views and Beyond”, Addison Wesley, 2002.

[25] D. Norris, “Communicating Complex Architectures with UML and the Rational ADS”, Proc. of the IBM Rational Software Development User Conference, 2004.

[26] D. Greefhorst, H. Koning and H. van Vliet, “The Many Faces of Architectural Descriptions”, Information Systems Frontiers, vol. 8, Issue no. 2, 2006, pp 103-113.

[27] S. Roselin Mary and P. Rodrigues,” Software Architecture- Evolution and Evaluation”, (IJACSA) International Journal of Advanced Computer Science and Applications, vol. 3, no.8,2012.

 ISBN: 978-1-61208-367-4

Challenges of Adopting Software Reuse: Initial Results

Sajjad Mahmood
Department of Information and Computer Science
King Fahd University of Petroleum and Minerals
Dhahran, 31261, Saudi Arabia
smahmood@kfupm.edu.sa

Ali Al Zayer
Department of Information and Computer Science
King Fahd University of Petroleum and Minerals
Dhahran, 31261, Saudi Arabia
g200163270@kfupm.edu.sa

***Abstract*—A significant number of software development organizations have started adopting software reuse in order to facilitate achieving quality software, faster and at a lower cost. Software reuse helps organizations to leverage the benefits of systematic reuse with respect to architecture, design, source code and testing artifacts. One of the major issues is that many organizations endorse software reuse prior to understating and testing their readiness for the reuse based development processes. The objective of this paper is to identify challenges associated with software reuse in an organization. We have performed a Systematic Literature Review (SLR) by applying customized search strings derived from our research question. We have identified challenges, such as domain analysis and modeling, lack of reuse skills and knowledge, lack of management support, high reuse cost and lack of component repositories as key challenges in software reuse. Our ultimate aim is to develop a model in order to measure organizations' readiness for software reuse activities.**

Keywords-systematic software reuse; challenges and barrier; systematic literature review; empirical studies.

I. INTRODUCTION

Software industry has been of the view that software artifacts can be reused to develop new applications. Software processes and artifacts have been reused since the early days of computing, as software reuse has the potential benefits to reduce development effort, reduce process risks and increase product quality and standard compliance [1]-[5].

Software development paradigms such as component-based development and service-oriented development encourage software reuse by supporting development based on reusable blocks of source code. In addition to source code, reuse of requirements patterns [6], system architecture, design and testing artifacts also have the potential to help achieve benefits associated with systematic reuse of software artifacts.

In addition to reuse benefits, numerous problems have been reported in the reuse initiatives [7][8]. Software reuse poses certain strategic challenges with respect to difficulty in maintaining a library of reusable artifacts and the cost of locating and adapting reusable artifacts [9]. Despite the importance of this problem, little research has been carried out to improve organizations' for adopting software reuse based development process. Understanding issues related to organizations readiness can help to ensure the successful outcome of projects.

In this paper, we aim at identifying the challenges via a systematic literature review that impact software reuse. Identifying these challenges will help software organizations in addressing them and be ready for systematic reuse. Our long term research objective is to develop a software reuse readiness framework to assist software developers in measuring and improving their software reuse readiness prior to adopting reuse driven development paradigms. To do this, we intend to address the research question as follows:

RQ: What are the challenges associated with adopting software reuse?

The rest of this paper is organized as follows: Section II presents motivation of the paper. Section III presents the background. In Section IV, we present the research methodology and Section V discusses the initial results. We conclude the paper and discuss future work in Section VI.

II. MOTIVATION

Sherif and Vinze [15] presented a qualitative study based on a series of five cases to explore the individual and organizational barriers associated with the adoption of reuse. The study indicates that barriers to adoption of software reuse occur at both the individual and organization level. Mellarkod et al. [17] identified and assessed factors that influence developers' intention to reuse software assets. The study identified development of an infrastructure, self-efficacy and reuse experience as key factors that motivate individual developers to adopt software reuse. Similarly, Lucredio et al. [16] used survey based approach involving Brazilian organizations to identify some of the key factors in adopting an organization wide software reuse program.

 ISBN: 978-1-61208-367-4

To the best of our knowledge, no explicit SLR-based empirical study has been conducted to identify the challenges associated with adopting software reuse in an organization. The initial results of this study are important for both practitioners and researchers to better understand the current state-of-the-art literature in the context of adopting systematic reuse. This study uncovers the challenges that need better management during adoption of software reuse in a given organization.

III. BACKGROUND

Software developers have reused abstractions and process ranging from objects to commercial off the shelf components. Over the last couple of decades, a number of software reuse focused development paradigms have evolved, such as component-based development [18][19], software product lines [20], etc. Figure 1 presents a summary of reuse driven software development paradigms.

However, a significant number of software products developed using these paradigms have faced problems due to insufficient preparation and poor management both by reusable code developers and component integrators.

Understanding issues related to organization's software reuse will help ensure the successful outcome of projects. Hence, in this paper, we conduct a systematic literature review to identify challenges associated with adopting software reuse during development of software applications. The collected data focuses on challenges for effective management of software reuse driven development processes.

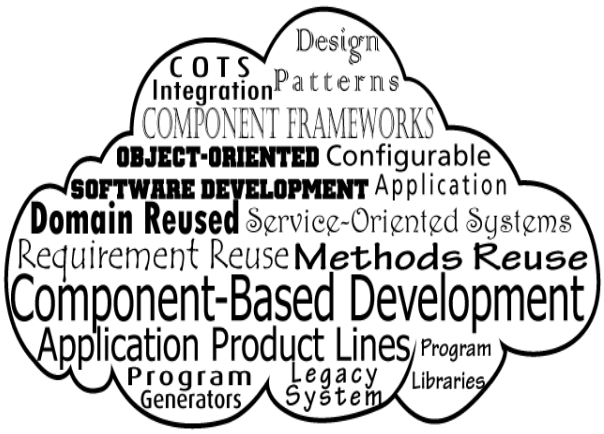


Figure 1. Software reuse paradigms.

IV. RESEARCH METHODOLOGY

In this study, we aim at identifying the challenges associated with software reuse during development of a software application. In order to address the research question in hand, the Systematic Literature Review (SLR) [10] was used as a tool for data collection. SLR is a well-established empirical study technique for identifying, assessing and analyzing published studies to investigate a specific research question. Furthermore, the SLR approach provides a higher level of validity in its findings. A systematic review protocol was prepared to outline the review process. The main phases [11][12] in our research methodology are as follows:

- Construct search strategy and search relevant articles.
- Carryout the study selection process.
- Apply study quality evaluation.
- Extract and analyze the data.

In this paper, we focus on the challenges associated with adopting software reuse in an organization. In order to address that, we are going to address the following research question:

RQ1: What are the challenges of software reuse in developing projects?

The search strategy used is based on the following steps:

a) Derive the main terms from Population, Intervention and Outcome.
b) Find the synonyms and of the derived terms obtained in the first step.
c) Validate these terms in various academic databases
d) AND operator is used to connect main terms (if allowed depending on the academic databases).
e) OR operators, is used to connect synonyms and similar spellings. (If allowed academic databases).

Based on our search strategy, we have come up with the following search terms:

- POPULATION: software reuse
- INTERVENTION: project development challenges and barriers.
- OUTCOME OF RELEVANCE: challenges and barriers in project development of software reuse.
- EXPERIMENTAL DESIGN: SLRs, case studies, empirical and theoretical studies, researchers and expert opinions.

After testing our main terms in several academic databases, the most relevance terms used to the topic are as follows:

 ISBN: 978-1-61208-367-4

- Software reuse: "Software reuse" OR "architecture reuse" OR "component reuse" OR "reuse environment" OR "product line based reuse".
- Software Development: "Software Development" OR "Software Implementation" OR "Software Coding".
- Challenges: "Challenges" OR "problems" OR "difficulties" OR "complications" OR "obstacles" OR "barriers" OR "risks".

The final search string has been designed after trail search, which is as follows:
(("Challenges" OR "problems" OR "difficulties" OR "complications" OR "obstacles" OR "barriers" OR "risks") AND
("Software reuse" OR "Architecture Reuse" OR "Component Reuse" OR "Code Reuse" OR "Product Line Based Reuse" OR "Software Reusability")
AND
("Software Development" OR "Software Implementation" OR "Software Application"))

Our focus was based on the following digital library:

- IEEE Explore. (http://ieeexplore.ieee.org)

The following inclusion criteria were used:

- Conference proceedings, magazines and journals published after 1980.
- Papers published in any of the primary or secondary resources mentioned previously.
- Studies which focus on answering our research question.

The following exclusion criteria were used:

- Duplicated or repeated studies.
- Manuscripts written in a language other than English language.
- Technical reports and white papers.
- Graduate projects, Master theses and PhD dissertations.
- Textbooks, whether in print or electronic.

For any paper to pass the initial phase, a quality assessment was done and four quality criteria were defined, as shown in Table I. We have selected 36 articles which meet the inclusion and quality criteria. Next, we extracted data from the final selected papers to address our research question. Table II presents the data extracted from the selected articles.

TABLE I. QUALITY ASSESSMENT

Criteria	Notes
Are the findings and results clearly stated in the paper?	Yes =1 No =0
Is there any empirical evidence on the findings?	Yes =1 No =0
Are the arguments well- presented and justified?	Yes =1 No =0
Is the paper well referenced?	Yes =1

TABLE II. DATA EXTRACTION FORM

Extracted Data	
▪ Publication Name	▪ Author(s)
▪ Publication Date	▪ Geographical Location
▪ Reference Type	▪ Publication Type
▪ Publisher	▪ Challenges

V. INITIAL RESULTS AND DISCUSSION

In this paper, we report our initial results based on IEEE electronic database. Table III shows the total number of results retrieved from IEEE electronic database. After initial round of screening by reading the title and abstract, 73 articles were selected. Next, full text of the 73 articles was read and 36 primary studies were finally selected.

TABLE III. SEARCH EXECUTION

Resource	Total Results	Initial Selection	Final Selection
IEEExplore	1395	73	36

In our SLR, we have classified the papers found into seven study strategies, which are commonly used in the empirical software engineering, as shown in Table IV. The majority of the selected articles used case study research method.

TABLE IV. STUDY STRATEGIES USED

Study Type	Count
Case Studies	24
Interviews	1
Experience Report	3
Systematic Literature Reviews	0
Survey/Questionnaire	6
Delphi Study	2
Total	**36**

Table V shows the country-based analysis for the papers included in the SLR study. Twenty studies were carried out in USA, three each in China and Spain, and two in Canada and United Kingdom, respectively.

 ISBN: 978-1-61208-367-4

TABLE V. STUDY COUNTRIES

Country	Count	Country	Count
Canada	2	Saudi Arabia	1
China	3	Spain	3
Germany	1	Italy	1
Japan	1	United Kingdom	2
Malaysia	2	USA	20

To answer our research question, the data were extracted and synthesized from the 36 finally selected studies. We have identified eight challenges for systematic reuse during development of software applications, as shown in Table VI.

TABLE VI. LIST OF CHALLENGES

Challenges	Freq. (n=36)	%
Domain analysis and modeling	29	83
Lack of reuse skills and knowledge	27	75
Lack of management support	12	33
High reuse cost	12	33
Lack of component storage	9	25
Lack of documentation	7	20
Lack of proper IT infrastructure	4	11
Lack of team awareness	2	6

In our study, the most common software reuse challenge is ‘domain analysis and modeling’ (83%). The fact that in software reuse-driven development, practitioners need to carry out detailed domain analysis and modeling to search and select suitable reusable components. The second highest ranked challenge is ‘lack of reuse skill and knowledge’. For example, Gonzalez [13] identified that the users of object oriented software components face cognitive gap in knowledge and often face difficulty in understanding the vocabulary used in component documentations.

About 33% of the articles in our study described ‘lack of management support’ and ‘high reuse cost’ as another major challenges. ‘Lack of component storage’ has been mentioned in about 25% of the articles. The main reason for this challenge is lack of standard reuse environments [9] and repositories. Furthermore, ‘lack of documentation’ has also been an important challenge in reuse based development. For example, Mahmood and Khan [14] empirical study indicates that the lack of good component documentation presents a risk for use of reusable components.

VI. CONCLUSION AND FUTURE WORK

Systematic software reuse facilitates achieving quality software, faster and at a lower cost. Despite the potential benefits associated with software reuse, software organizations struggle with adopting reusable components during development of a software application. Due to availability of a number of reuse-driven development paradigms and the increasing trend of adopting reusable components, we aim to discover challenges associated with systematic reuse.

In our initial results, the frequently mentioned challenges for systematic reuse are domain analysis and modeling, lack of reuse skills and knowledge, lack of management support, high reuse cost and lack of component repositories.

As part of future work, we plan to carry out SLR in other major databases, namely, ACM, Science Direct, Springer Link, and John Wiley. We also plan to identify solutions, in the form of best practices, for each of the frequently mentioned challenge. We intend to find the best practices by carrying out an empirical study with software industry.

ACKNOWLEDGMENT

The authors would like to thank King Fahd University of Petroleum and Minerals (KFUPM) for its continuous support of research.

REFERENCES

[1] W. B. Frakes, "Software reuse research: status and future", IEEE Trans. Softw. Eng., vol. 31, pp. 529-536, 2005.

[2] W. B. Frakes and S. Isoda, "Success factors of systematic reuse. Software", IEEE Software, vol. 11, pp. 14-19, 1994.

[3] A. Mili, S. Yacoub and H. Mili, "Toward an engineering discipline of software reuse", IEEE Software, vol. 16, pp. 22-31, 1999.

[4] L. Prechelt, B. Unger, W. F. Tichy and P. Brossler, "A controlled experiment in maintenance comparing design patterns to simpler solution", IEEE Trans. Softw. Eng., vol. 27, pp. 1134-1144, 2001.

[5] S. Watanabe, "Professionalism through OO and reuse", IEEE Software, vol. 14, pp. 26, 1997.

[6] X. Franch, "Software requirement patterns", 35th International Software Engineering Conference, pp. 1499-1501, 2013.

[7] D. Card, and E. Comer, "Why do so many reuse programs fail", IEEE Software, pp. 114-115, 1994.

[8] M. Jha, and L. O'Brien, "A comparison of software reuse in software development communities", 5th Malaysian Software Engineering Conference, pp. 313-318, 2011.

[9] S. Mahmood, M. Ahmed, and M. Alshayeb, "Reuse envirnoments for software artifacts: analysis framework", 12th IEEE/ACIS International Conference on Computer and Information Sciences, pp. 35-40, 2013.

[10] B. Kitchenham, and C. Charters, "Guidelines for performing systematic literature reviews in software engineering", Keele University and Durham University Joint Report - EBSE 2007-001, EBSE 2007-001, 2007.

[11] S.U. Khan, M. Niazi, and R. Ahmad, "Factors influencing clients in the selection of offshore software outsourcing vendors: an exploratory study using a systematic literature review", Journal of Systems and Software, vol. 84, pp. 686-699, 2011.

[12] M. Niazi, S. Mahmood, M. Alshayeb, A. M. Qureshi, K. Faisal and N. Cerpa, "Towards identifying the factors for project management success in global software development:

 ISBN: 978-1-61208-367-4

Initial results", The Eigth International Conference on Software Engineering Advances, pp. 285-290, 2013.

[13] P. A. Gonzalez, "Applying knowledge modelling and case-based reasoning to software reuse", IEEE Software Proc., vol. 147, pp. 169-177, 2000.

[14] S. Mahmood, and A. Khan, "An industrial study on the importance of software component documentation: A system integrator's perspective", Information Processing Letters, vol. 111, pp. 583-590, 2011.

[15] Karma Sherif and Ajay Vinze, "Barriers to adoption of software reuse: A qualitative study", Information and Management, vol. 41, pp. 159-175, 2003.

[16] Lucredio et al., "Software reuse: the Brazilian industry scenario", Journal of Systems and Software, vol. 81, pp. 996 - 1013, 2008.

[17] V. Mellarkod, R. Appan, D. R. Jones and K. Sherif, "A multi-level analysis of factors affecting software developers' intention to reuse software assests: an empirical investigation", Information and Management, vol. 44, pp. 613 - 625, 2007.

[18] M.A. Khan and S. Mahmood, "A graph based requirements clustering approach for component selection", Advances in Engineering Software, vol. 54, pp. 1-16, 2012.

[19] S. Mahmood, R. Lai and Y.S. Kim, "Survey of component based software specification", IET Software, vol.1, pp. 57-66, 2007.

[20] F.V. Linden, K. Schmid and E. Rommes, "Software product lines in action: The best industrial practice in product line engineering', Springer, 2007.

 ISBN: 978-1-61208-367-4

Comparison of Stakeholder Identification Methods – The Effect of Practitioners Experience

Markus Kelanti, Samuli Saukkonen
Department of Information Processing Science
University of Oulu
Oulu, Finland
{markus.kelanti; samuli.saukkonen}@oulu.fi

***Abstract*—Stakeholder analysis is an important part of Requirements Engineering activities. Since stakeholders affect, and can be affected by, a system under development, it is important to identify them. While several stakeholder analysis methods are available, there has been less discussion about their effectiveness when practitioners have different levels of work experience. This paper evaluates how the stakeholder identification method affects the amount and variation of stakeholders in cases where practitioners have relevant, not relevant or no experience at all. The research investigated this question by conducting a study in a university Requirements Engineering course comparing three different stakeholder identification methods where participants' work experience was known. This paper discusses the results of the experiment and their implications. The results highlight the importance of relevant experience and systematic approach to stakeholder identification.**

Keywords-Stakeholder; Stakeholder identification.

I. INTRODUCTION

Stakeholder analysis is considered an important part of Requirements Engineering (RE). Identifying different types of stakeholders is crucial to creating successful software projects and is recognised by initiatives like IEEE standard 830 [2] and SWEBOK [1]. While many papers emphasise the importance of stakeholders, the identification process itself is not as well defined or documented in the RE literature [5][7]. Several authors, e.g., Sharp et al. [5] and McManus [6], criticise the lack of clear and efficient methods for identifying actual stakeholders.

This problem has gained some attention, and several concrete methods [5][6] have been developed to conduct the analysis, including the identification of the actual stakeholders. StakeNet [8], in addition to providing a stakeholder identification method, also studied the effectiveness of the method. Since the problem is gaining attention, an interesting question arises as to how the effectiveness of different stakeholder identification methods is affected by practitioners' experience. In other words, what is their ability to produce a list or group of stakeholders if different methods are used? The goal of this paper is to answer the following research question:

How the use of a stakeholder identification method does affect the effectiveness of the stakeholder identification process for experienced and inexperienced practitioners?

In this paper, effectiveness is defined as how fast a list of stakeholders can be generated for a single system. The three methods used in this study represent three different approaches to stakeholder identification in order to determine whether the identification results are different. The following stakeholder identification methods were used: a systematic approach from Sharp et al. [5], a question-based approach method used by McManus [6] and a general list of possible stakeholders that should be considered when developing software systems, from Lauesen [12]. In order to answer the research question, a study was conducted in a university RE course. The results of the study were analysed to determine whether a specific method had any advantage. In the study, the level and quality of the students' experience was controlled to determine the role of experience in the results.

The rest of the paper is structured as follows. Section 2 describes the literature regarding stakeholders and how the concept is generally used in RE. Section 3 describes the stakeholder identification methods used in this study and how the study was conducted. Section 4 presents and discusses the results. Section 5 discusses limitations and possible threats to the validity of the findings, and Section 6 provides a conclusion and future directions for research on this topic.

II. LITERATURE

The concept of the stakeholder was popularised by Freeman [9]. Freeman described a stakeholder as a group or an individual who is affected by the achievement of an organisation's objectives or who can affect on them. Stakeholders and stakeholder analysis was first used mostly in management literature and practice to understand the different stakeholder needs in a company [10]. Eventually, the concept made its way to RE.

In RE, a stakeholder can be identified as a person or a group who will be affected by the system either directly or indirectly [11]. Sometimes, there is no clear definition of a stakeholder; instead they are specific groups of people who make demands of a particular system [12]. Depending on the development domain and target market, common stakeholders include various end users, customers, engineers and managers [3][11]. In general, stakeholders are considered persons, groups, or organisations that express needs regarding a particular system, are affected by it, or can somehow affect it.

The importance of stakeholders in RE is most visible in the elicitation process. Stakeholders are one of the main sources of information in the elicitation process that creates actual requirements. However, stakeholders often have conflicting

 ISBN: 978-1-61208-367-4

views and needs, are unable to express their needs in a detailed manner or they demand a solution that does not match their real need. Even if this makes requirements analysis a hard and tedious task, identifying and understanding stakeholders' needs more comprehensively and reaching consensus among them increases the quality of the software product [3][11][12].

Clearly, identifying stakeholders and analysing their needs is an important task. However, the process of stakeholder identification is often an ad-hoc analysis or left for the practitioners to figure out for themselves. The literature has criticised this lack of guidance for practitioners [5][6][8]. Stakeholder identities are either assumed to be obvious to the practitioners or to fit into categories too broad and generic to be useful. Some studies just present lists or categories of identified stakeholders, such as the most commonly known clients/customers, users and developers [3][11][12]. Other papers expand these lists by adding businesses, projects and products [13][14][15] or government agencies [16], organisations [16] and the general public [17] as stakeholders. Pachecho and Garcia [18] systematically surveyed the contemporary literature for state-of-the-art identification methods and concluded that the stakeholder identification process still lacks standards and proper guidance.

Several authors have addressed the above criticisms by developing concrete methods to aid in the identification process. For example, Lyytinen and Hirschheim [19] provide some guidance in identifying stakeholders; they note that identification itself is far from trivial. Further, McManus [6] uses the question list provided by the World Bank and criticises the lack of exact methods to identify concrete stakeholders. Similarly, Sharp et al. [5] present a systematic approach for identifying stakeholders in the absence of a clearly defined identification method. The latest advances include social media applications like StakeNet [8], a stakeholder identification method based on social networks. In this method, stakeholders are first identified by asking a person to identify an initial set of stakeholders. These stakeholders are then asked to produce another set of stakeholders, and the pattern repeats itself until a stable network of interconnected stakeholders is formed.

III. RESEARCH SETTING

The experiment ran as part of an RE course at the University of Oulu. In order to obtain the necessary data to answer the research question, basic software experiment guidelines [20][21] were followed in designing the study. This section describes the stakeholder identification methods, research setting, execution and how the data were analysed.

A. Experiment setting

The RE course is a part of the 3rd year Bachelor's degree studies in Information Systems and Software Engineering (SE) and is compulsory for every student of the program. One topic in the course is stakeholder analysis as a part of RE activities. The experiment was designed to be the compulsory practice session necessary for every student to pass the course. The students, both Finnish and foreign, were all from the same university and department. The majority of the students were Finnish.

To answer the research question, an experimental setup comparing three different stakeholder identification methods was conducted. Before the experiment, students completed a background questionnaire about their experience. This questionnaire asked students about their work experience, specifically whether the experience was generally related to SE, and how many total years of experience they had. Experience was divided into SE and other experience, since students might be experienced in other fields as well. This information was used to split the students into three different groups: those with experience in SE, those with no previous experience in SE and those who had related experience but not in SE.

The scenario used in the experiment required the students to develop a requirements document for the new department timetable software named LUKKARI. The scenario stated that the old timetable software was unsuitable for today's needs and should therefore be replaced with a new system. This scenario was selected and developed to ensure that each student understood how a timetable system works since they had been using one during their studies. This was also done to avoid situations where some students would not have a specific domain expertise that could affect the results. The minimum basic functionalities of LUKKARI allow users to:

- Log in and out
- Browse their own and course timetables
- Create, edit and remove items to their own timetable
- Access the timetable through a web browser
- Add, edit and remove resources from timetable items

In order to obtain the stakeholder data, an answer sheet was designed for the students, which asked them to name any identified stakeholder, give a short description and provide a rationale for why the student thinks that stakeholder is relevant. In addition, researchers recorded the time when the answer sheet was returned to calculate the amount of time used to identify stakeholders.

Students were informed that their answers would not affect their course scores. However, they were told that they could use their own results when they began working with the Requirements Specification documents required by the course. This helped to remove possible pressure from the students while also providing an incentive to identify stakeholders.

B. Methods Under Experiment

The experiment was designed to present two different types of identification methods: McManus's questionnaire method (based on World Bank's stakeholder analysis) [6], and Sharp et al.'s systematic analysis [5]. It should be noted that only the stakeholder identification part was used from both methods. The control method was based on an analysis process described in Lauesen's textbook [12] because it was already part of the course. These methods fit the restrictions of the experiment because all material had to be in a written format, and all methods had to be designed for or used in SE.

1) Systematic method

The systematic method of Sharp et al. [5] for identifying stakeholders uses four baseline stakeholder groups: users, developers, legislators and decision makers. Users are the people, groups or companies who interact, directly control or use the software. Developers have a stake in the system's final requirements specification but are not themselves users. Legislators are, for example, government agencies, trade unions and legal representatives, all of which act nationally and

 ISBN: 978-1-61208-367-4

internationally, setting guidelines for operations that affect the product's development or its final use. Finally, decision makers direct both development and user organisation. The identification method itself is straightforward:

1. Identify all specific roles within the baseline stakeholder group.
2. Identify supplier stakeholders for each baseline role. The supplier stakeholders provide information or supporting tasks for the baseline stakeholders.
3. Identify client stakeholders for each baseline role. The client stakeholders process or inspect the products of the baseline stakeholders.
4. Identify satellite stakeholders for each baseline role. The satellite stakeholders interact with the baseline stakeholders in a variety of ways.
5. Repeat steps 1 to 4 for each of the stakeholder groups identified in steps 2 to 4.

2) Questionnaire method

The questionnaire method represents the question-based stakeholder identification method used by McManus [6]. Compared to the systematic method, the questionnaire method does not provide any systematic way to address stakeholder categories, presenting only a pre-defined list of questions that can reveal stakeholders. McManus uses a question list developed by the World Bank to identify stakeholders in pre-defined categories. These questions are [6]:

1. *Who might be affected (positively or negatively) by the development concern to be addressed?*
2. *Who are the "voiceless" for whom special efforts may have to be made?*
3. *Who are the representatives of those likely to be affected?*
4. *Who is responsible for what is intended?*
5. *Who is likely to mobilize for or against what is intended?*
6. *Who can make what is intended more effective through their participation or less effective by their non-participation or outright opposition?*
7. *Who can contribute financial and technical resources?*
8. *Whose behaviour has to change for the effort to succeed?*

3) Control method

The control method, which is derived from the course textbook [12], is a simple list of very specific definitions for identifying stakeholders. As all the participating students are familiar with this list, it provides an ideal control method. In addition, it roughly follows the same manner of describing stakeholders as other undergraduate-level SE textbooks.

The control method defined stakeholders mainly as people who are needed to ensure the success of a project, who can be [12]:

1. *The sponsor who pays for the product. He wants value for his money.*
2. *Daily users from various departments. They have to live with the product and, without their support, there will be no success.*
3. *Managers of the departments. They want business advantages from the system.*
4. *The company's customers (clients of the system). Often they will see changes too, and without their support there will be no business advantages.*
5. *Business partners, for instance suppliers, carriers, and banks. If they will see changes, their support is essential too.*
6. *Authorities, for instance safety inspectors, auditors, local government.*
7. *IT people and hotline staff in case the product is to be developed in-house.*
8. *Other people providing resources for the product.*
9. *The daily users of the product at the client's site.*
10. *Managers and sponsors at the client's site.*
11. *IT people at the client's site.*
12. *Distributors and value-adders for our product. (Value adders – or VARs – may for instance be software houses that combine our product with other products or services.)*
13. *Competitors. They are definitely influenced by the product, but usually in an adverse manner. If so, they will not be treated as stakeholders. However, in some cases you depend on their co-operation, for instance if you are going to exchange data with them electronically. These situations may be delicate, and your best change is to create a win-win situation where they benefit too.*

C. Experiment execution

Before the actual session, students were divided into three groups based on their answers from the pre-questionnaire regarding their prior experience. The groups were balanced to include only students with relevant experience, students with no relevant experience or students with no experience at all. Students in these groups were then randomly assigned to one of the three stakeholder identification methods. Each student received a package containing instructions, an answer sheet and a description of the stakeholder identification method.

All students were required to participate in a 2-hour practice session. At the beginning, students were asked to pick up the answer sheet with their name on it and sit down to wait for the session to start without looking at the papers. Students were given a short 15-minute introduction to the experiment and were allowed to ask questions and clarifications about the experiment. Students were told to fill in the answer sheet according to the instructions and return it to the researchers when completed. No other time limit was imposed aside from the maximum 2 hours reserved.

Students were instructed to work alone, and the researchers supervised the session to enforce this rule. In addition, no computers were allowed, but only the given material and writing equipment. Two researchers were constantly present during the session to answer questions and ensure that the rules and instructions were followed.

All answer sheets were returned to the researchers after students completed them. Each answer sheet was then transferred to an Excel file to conduct the analysis.

 ISBN: 978-1-61208-367-4

D. Data analysis

Before analysing the results, the data needed to be checked for:

- Possible duplicates in case the same stakeholder was mentioned twice in one answer sheet. Duplicates were simply removed.
- A list of stakeholders that appeared as a single stakeholder in the answer sheet. In this case, the stakeholders were marked as separate stakeholders in the answer sheet.
- Plural or single stakeholder terms. All stakeholders in single terms were changed to plural.
- All stakeholder names were in English. In case a name was not given in English, two researchers agreed on the translation that matched the original name as precisely as possible.
- Extra lines or words. Only named stakeholders were considered as such; all other answers were removed, such as 'etc.', 'and so on' and '…'.

After the pre-analysis, the next step was to check how many unique stakeholders were identified. For this step, two researchers checked and compared each identified stakeholder against the others to determine whether they were exactly the same. All stakeholders were considered unique by default, and stakeholders were only considered the same if the name contained a clear typo, the stakeholder description was the same or it was otherwise obvious that the stakeholder itself was exactly the same. The most typical case was that identified stakeholders belonged to the same group but were miscategorised as a sub group or an individual stakeholder within the group. In this case, the stakeholders were considered unique.

Finally, two researchers worked together to evaluate whether the identified stakeholders were relevant stakeholders for the LUKKARI system. The main criterion for determining whether a stakeholder was relevant was the rationale provided for each stakeholder. Stakeholders were excluded from the study if the rationale was not provided or if it clearly indicated that the stakeholder was not connected to the described LUKKARI system.

Time was also measured to determine whether there was any significant difference in analysis time between different student groups or methods.

IV. RESULTS

In total, 51 students participated in the experiment and identified a total of 449 stakeholders, an average of 8.8 stakeholders per student. The results are shown in Table 1. The control method produced a total of 128 stakeholders, averaging 8 stakeholders per student.

There were 54 unique stakeholders, an average of 3.4 per student. These results include all three groups of students and form the baseline performance for comparing the performance of the two stakeholder identification methods. Given the results in Table 1, both the questionnaire and systematic methods outperform this baseline. The questionnaire method produced a slightly better average when comparing identified stakeholders per student while the systematic method was clearly better than the other two. Similar results were found with the unique stakeholders, the questionnaire being slightly better than the control, while the systematic method was superior overall.

Comparing the methods when the students were evenly distributed based on their experience, the questionnaire method was slightly better than the control method, but the systematic method clearly outperformed the others. The results are also similar with unique stakeholders, with the systematic method again outperforming the other two. This suggests that instead of relying on questions and categories, a systematic approach more accurately finds stakeholders and identifies unique stakeholders. Comparing the time spent identifying stakeholders, there is only a slight difference between the control and systematic methods. The questionnaire method took the most time of the three, so in this regard it was less effective.

When experience is measured separately, the students without experience produced a total of 223 stakeholders, with an average of 8.3 stakeholders identified per student. Of all stakeholders, 87 were unique stakeholders, an average of 3.1 unique stakeholders per student. This group forms the baseline for measuring the influence of experience. Table 1 clearly shows that experienced students performed better than the baseline in terms of the average number of stakeholders and unique stakeholders identified per student. The group with no relevant experience performed slightly poorer in regard to average stakeholders per student than the group with no experience at all. However, the totally inexperienced group identified more unique stakeholders.

TABLE I. OVERALL RESULTS FROM THE EXPERIMENT

	Control method	**Questionnaire method**	**Systematic method**
Students:	16	19	16
Total identified stakeholders:	128	165	156
Average per student:	8.0	8.7	9.8
Average time per stakeholder	4 min, 44 sec	5 min, 18 sec	4 min, 52 sec
Total unique stakeholders:	54	74	84
Unique stakeholders per student:	3.4	3.9	5.3
	No experience	**Experience but not SE**	**Experience from SE**
Students:	28	10	13
Total identified stakeholders:	223	72	141
Average per student:	8.3	7.4	10.8
Average time per stakeholder	5 min, 29 sec	5 min, 19 sec	4 min, 1 sec
Total unique stakeholders:	87	39	81
Unique stakeholders per student:	3.1	3.9	6.2

 ISBN: 978-1-61208-367-4

The experienced group clearly outperformed the other two groups. Experienced students were able to find more stakeholders than students with no experience. The groups with no experience at all and without relevant experience provided fewer stakeholders. What was surprising, however, was that the group that lacked relevant experience managed to find fewer stakeholders than that with no experience at all. Although the group still came up with more unique stakeholders, their experience from different domains might have caused this interference. While this study cannot provide an answer for this finding, it might be useful to experiment to investigate how the different experience affects the identification process. Finally, accounting for the time spent to identify a single stakeholder, there was only a slight difference between those with no relevant experience and those with no experience at all. Those with experience, however, were clearly faster. These results indicate that experience is a key attribute for identifying unique stakeholders, and relevant expertise provides a clear benefit.

TABLE II. AVERAGE OF IDENTIFIED STAKEHOLDERS IN EACH METHOD SEPARATED BY EXPERIENCE GROUPS

	Control method	**Questionnaire method**	**Systematic method**
Experienced	8.8	9.4	14.6
No relevant experience	7.3	6.8	7.7
No experience	7.1	8.3	8.4

The results were also tabulated according to the identification method group based on experience level, as shown in Table 2. While the comparison groups were clearly smaller, the use of a specific method still provided better results. The data shows that students benefitted from the method regardless of experience. The only difference was that the group with no relevant experience that specifically used the questionnaire method identified the least stakeholders. It should be noted that while the difference is not big, the group with no relevant experience showed the least amount of improvement over the other two groups. This result is interesting because general experience should indicate more information about stakeholders, whereas this experiment hints that specifically relevant experience matters more.

The rate at which sets of stakeholders occurred in the different method and experience groups were evaluated, as shown in Table 3. Comparing the frequency of stakeholders found no clear difference between any of the groups. The main stakeholders each group identified were similar in kind; generally, the stakeholders were organisational units of the university, stakeholders related to the university itself and stakeholders related to the LUKKARI system. Mainly, the frequency of their appearance varied. When all stakeholders were ranked according to occurrence, each group received similar results. While some groups' position in the list varied greatly, no group clearly appeared more frequently in one group and less in another. This indicates that identification method and experience did not bias students to select particular stakeholders that would greatly differ from other groups.

V. THREATS TO THE EXPERIMENT'S VALIDITY

Students were expected to do the work individually in order to test whether the method actually helps individual students identify stakeholders. Communication between students and data searches were deliberately denied to control the experiment. In real life, however, work is often done in teams, and several people can work on the same task. In addition, access to company resources and the Internet also provide resources to help the identification process. Therefore, this study cannot be directly compared to a real environment as such.

The study did not consider how valid and important each stakeholder was for the system. This was intentionally excluded because determining validity and importance was beyond the scope of this study. The study concentrated only on determining which identification method is more likely to produce a larger and more accurate set of stakeholders, compared to working without any specific method at all.

TABLE III. COMMON STAKEHOLDERS

Control method	**Count**	**Questionnaire method**	**Count**	**Systematic method**	**Count**
IT Services	10	Students	10	Students	27
Students	9	University of Oulu	5	Student Councilors	15
Teachers	7	Teachers	5	LUKKARI Developer	12
Student Councilors	7	IT Services	5	University of Oulu	11
LUKKARI Developer	6	Student Councilors	5	IT Services	11
University of Oulu	5	Lukkari Developer	5	Teachers	10
Ministry of Education	3	University of Oulu Management	3	LUKKARI Development Team	7
External Consults	3	Course Management System Developers	2	University of Oulu Management	7
Requirements Engineers	3	Department Managers	2	Project Financiers	6
LUKKARI Development Team	3	LUKKARI Administrators	2	Teaching Staff	5
No experience	**Count**	**Experience but not SE**	**Count**	**Experience from SE**	**Count**
Students	17	Students	14	Students	15
Student Councilors	13	Teachers	9	IT Services	10
IT Services	10	LUKKARI Developer	7	LUKKARI Developer	9
Teachers	9	IT Services	6	Student Councilors	8
University of Oulu	8	Student Councilors	6	University of Oulu	8
LUKKARI Developer	7	University of Oulu	5	IT Support	4
Project Financiers	5	University of Oulu Management	5	University Financiers	4
University Administration	5	LUKKARI Development Team	5	Teachers	4
Ministry of Education	4	LUKKARI Administrators	4	University of Oulu Management	4
LUKKARI Development Team	4	LUKKARI Project Managers	3	Teaching Staff	3

 ISBN: 978-1-61208-367-4

No quantitative analysis was performed on the results due the nature of the study. The rationale behind the decision to use only a qualitative analysis was that the study was designed to be more explorative to see whether the methods provided clearly different results. Each answer provided by the students was therefore analysed separately to understand whether the stakeholders were the same, whether the stakeholder had a rationale to be a stakeholder for the LUKKARI system and what kind of stakeholder groups were formed by the different methods. Therefore, the quantitative analysis was used to gain an insight into whether experience and method had any effect. However, quantitative analysis could provide more insight about the results from this study. Based on the results of this study, a longer study with a larger audience should be conducted.

The experiment was limited only to students, which affects the generalisation of the results. However, this shortcoming was addressed by pinpointing students with relevant experience in software engineering and classifying them as a separate group for analysis. Experimenting in a real development situation should be the next step after this experiment to confirm the large-scale effect of experience.

VI. CONCLUSION AND FUTURE WORK

The current advances in the development of stakeholder identification methods are gaining more attention, and defined stakeholder identification methods for RE have already been published. While the need for these methods is receiving more attention, comparing their effectiveness with practitioners having different types of expertise is less studied. This paper contributes to this issue with an experiment in which different stakeholder identification methods, the systematic method of Sharp et al. [5] and the questionnaire method of McManus [6], are measured against standard RE education literature guidance [12].

When the results from groups using either systematic or question-based stakeholder identification method were compared to a control group, both groups were able to identify more stakeholders than the control group. The results also indicate that the systematic identification method performed slightly better than the questionnaire. Based on this finding, a systematic stakeholder identification method provided more identified stakeholders, although a defined method, like a questionnaire, was found to be better than just a list of possible stakeholders.

The results show that experience is an important factor in stakeholder identification. The main finding was that experienced participants were able to identify more stakeholders than those without relevant experience or with no experience at all, regardless of what identification method was used. In addition, those without relevant experience actually performed slightly worse compared to others, indicating that the type of experience is also relevant. Using a defined stakeholder identification method in this study clearly increased the amount of stakeholders identified by both experienced and non-experienced participants.

One area for future work will be testing these methods with companies working with real customer projects and extending the experiment to determine whether identified stakeholders are actually important for a software product. Another research topic is to study how the guide helps to identify stakeholders and whether the efficiency of a single method depends on the application domain. In this regard, one direction is to analyse approaches like StakeNet [8], where several practitioners participate in the identification process to generate a richer set of stakeholders.

Finally, the quality and domain of the experience itself should be studied. This study demonstrated an interesting anomaly in the results between those that were experienced in SE and those without relevant experience. Therefore, one of the future research activities should concentrate on this particular finding.

ACKNOWLEDGMENT

The authors would like to thank the ITEA2 AMALTHEA and Digile N4S projects for providing support for the research.

REFERENCES

[1] IEEE. Guide to the software engineering body of knowledge, http://www.swebok.org [retrieved: August, 2014]

[2] IEEE. IEEE Recommended Practice for Software Requirements Specifications, IEEE Std. 830–1998. IEEE Press, 345 East 47th Street, N.J., 1998.

[3] A. Aurum and C. Wohlin, 'Engineering and managing software requirements', Springer-Verlag, Berlin, Heidelberg, 2005.

[4] I. Sommerwille and P. Sawyer, 'Requirements engineering – A good practice guide', John Wiley & Sons Ltd, Chichester, West Sussex, 1997.

[5] H. Sharp, A. Finkelstein, and G. Galal, 'Stakeholder identification in the requirements engineering process', Tenth International Workshop on Database and Expert Systems Applications (DEXA 99), Aug. 1999, pp. 387–391.

[6] J. MacManus, 'A stakeholder perspective within software engineering projects', IEEE International Engineering Management Conference, Vol. 2, Oct. 2004, pp. 880–884.

[7] C. Pacheco and I. Garcia, 'Effectiveness of stakeholder identification methods in requirements elicitation: Experimental results derived from a methodical review', Eighth IEEE/ACIS International Conference on Computer and Information Science, Jun. 2009, pp. 939–942.

[8] S. L. Lim, D. Quercia, and A. Finkelstein, 'StakeNet: Using social networks to analyse the stakeholders of large-scale software projects', Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE'10), May. 2010, pp. 295–304.

[9] R. E. Freeman, 'Strategic management: A stakeholder approach', Pitman, Boston, 1984.

[10] A. P. Friedman and S. Miles, 'Stakeholders, theory and practice' Oxford University Press, Oxford, 2006.

[11] I. Sommerville, 'Software engineering, 7th ed.', Pearson Education Ltd, Edinburgh Gate, Harlow, 2004.

[12] S. Lauesen, 'Software requirements: Styles & techniques', Pearson Education, 2002.

[13] S. Barney, A. Aurum, and C. Wohlin, 'Quest for a silver bullet: Creating software product value through requirements selection', Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications, Aug. 2006, pp. 274–281.

[14] S. Barney, C. Wohlin, H Ganglan, and A. Aurum, 'Creating software product value in China', IEEE Software, vol. 25, iss. 4, 2009, pp. 84–90.

 ISBN: 978-1-61208-367-4

[15] S. Barney, A. Aurum, and C. Wohlin, ‘A product management challenge: Creating software product value through requirements selection’, Journal of Systems Architecture, vol. 54, iss. 6, 2008, pp. 576 – 593.

[16] J. Fedorowicz, U. J. Gelinas Jr, J. L. Gogan, and C. B. Williams, ‘Strategic alignment of participant motivations in e-government collaborations: The internet payment platform pilot’, Government Information Quarterly, vol. 26, iss. 1, Jan. 2009, pp. 51–59.

[17] H. In and B. Boehm, ‘Using WinWin quality requirements management tools: A case study, Annals of Software Engineering, vol. 11, iss. 1, Nov. 2001, pp. 141–174.

[18] C. Pachecho and I. Garcia, ‘A systematic literature review of stakeholder identification methods in requirements elicitation’, Journal of Systems and Software, vol. 85, iss. 9, Sep. 2012, pp. 2171–2181.

[19] K. Lyytinen and R. Hirschheim, ‘Information systems failures - a survey and classification of the empirical literature’, Oxford Surveys in Information Technology, vol. 4, 1987, pp. 257–309.

[20] N. Juristo and A. M. Moreno, ‘Basics of software engineering experimentation’, Springer Publishing Company, 2010.

[21] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, ‘Experimentation in software engineering’, Kluwer Academic Publishers, Norwell, 2000.

 ISBN: 978-1-61208-367-4

What Are the Features of This Software? An Exploratory Study

Barbara Paech, Paul Hübner

University of Heidelberg
Institute of Computer Science
Heidelberg, Germany
{paech,huebner}@informatik.uni-heidelberg.de

Thorsten Merten

Bonn-Rhein-Sieg University of Applied Sciences
Dept. of Computer Science
Sankt Augustin, Germany
thorsten.merten@h-brs.de

***Abstract*—Application systems are often advertised with features, and features are used heavily for requirements management. However, often software manufacturers only have incomplete information about the features of their software. The information is distributed over different sources, such as requirements documents, issue trackers, user manuals, and code. In this paper, we research the occurrence of feature information in open source software engineering data. We report on a case study with three open source systems. We analyze what information about features can be found in issue trackers and user documentation. Furthermore, we study the abstraction levels on which the features are described, how feature information is related, and we discuss the possibility to discover such information semi-automatically. To mirror the diversity of software development contexts, we choose open source systems, which are quite different, e.g., in the rigor of issue tracker usage. The results differ accordingly. One main result is that the user documentation did not provide more accurate information than the issue tracker compared to a provided feature list. The results also give hints on how the management of feature relevant information can be supported.**

***Index Terms*—feature; requirements management; mininig software repositories; issue tracker; user documentation**

I. INTRODUCTION

In requirements management, features of application software are heavily used to package requirements. At least for the following three purposes: release planning in software product management [1], software product line engineering [2] and requirements feature interaction detection [3]. The corresponding approaches typically assume a dedicated feature and requirements representation. However, in industry features often are managed implicitly. Typically, they are used within a project to develop a part of a software product, but they are not collected in a dedicated document and maintained over time. The paper by Alspaugh and Scacchi shows that open source software (OSS) development projects typically do not have an explicit requirements or feature description and stipulates that this might also be true for many commercial software development projects [4]. In our work, we reported about feature knowledge being implicit in answers to requests for proposals [5]. We and others have reported on a heterogeneous requirements pool being the basis for release planning in industry [1], [6], [7]. Thus, in order to get a feature view of a software product it is often necessary to detect the features from data sources other than requirements or feature documents. Three feature-related information sources are typically available in software projects in industry:

- Bugs and feature requests in an issue tracker
- User documentation
- Code in a version control system

For feature location in code, typically the existence of documentation about features is assumed [8]. As we are interested in deriving the features, we focus on issue trackers (ITS) and user documentation (UD). UD has already been recommended as a substitute for a requirements specification by Dan Berry et al. in [9]. ITS are a well-known source for features, as often issues are explicitly tagged as features. However, feature tagging is not always reliable as has been shown by Herzig et al. in [10]. For example, they found that only 40% to 72% of Bugzilla issues are correctly classified as feature requests and many issues classified as bugs or improvements do actually contain feature requests.

Thus, it is necessary to analyze in more detail what information about features can be found in these sources. Although the Mining Software Repositories[1] community does some work about categorizing features and bug reports, there is no work identifying the individual feature descriptions in the ITS or UD data. The long term-goal of our research is to develop an approach to semi-automatically derive a feature representation from these data sources. As a first step, we present an explorative study analyzing the feature information of three different open source systems. The goal is to explore the kind and quality of feature information in ITS and UD.

The rest of the paper is structured as follows. Section II presents the planning and operation of the case study. Section III presents the results of the study. Related work is discussed in Section IV and an overall summary and outlook on future work is given in Section V.

[1] http://msrconf.org.

 ISBN: 978-1-61208-367-4

II. Case Study Planning and Operation

In this section, we describe the definition and planning of our study, the operation and the threats to validity.

A. Study Definition and Planning

We applied case study research, as this is an exploratory study trying to understand a real-world phenomenon [11]. The main research question is:

What information about software features can be found in the user documentation and issue tracker of a software product, and how well is this suited to derive a feature representation of the software?

This question is detailed into the following research questions:

- RQ1: What feature information can manually be derived from the issue tracker and the user documentation?
- RQ2: What are the commonalities and differences of feature information from UD and ITS and how well does the information fit to the feature list provided by the developers themselves?
- RQ3: How easily could this information be derived semi-automatically?

The study was conducted on open source project data, since it is most easily available. Based on prior experience, the projects were selected so that their combination fulfills the following criteria (see Table I):

- Availability of ITS and UD and of an explicit feature list compiled by the software developers themselves
- Different domains
- Different size of product and ITS and UD data
- Different user groups. We looked for projects with private users and/or professional users.
- Different kinds of ITS. Radiant uses a lightweight ITS (GitHub) with simple tagging possibilities to categorize issues. OFBiz uses an industry standard ITS (Jira) which supports multiple categorization options and status. Mixx uses a heavyweight ITS (LaunchPad) which additionally provides the option of connecting blueprints and user questions to the issues.
- Diversity of the quality of the provided information in the ITS. E.g., Mixxx uses the ITS systematically, whereas Radiant uses it ad-hoc.
- Different completeness of our analysis. The large projects could only be analyzed partially, but are more representative for the situation in industry.

TABLE I. Selection of OSS Projects

	Mixxx	OFBiz	Radiant
Domain	DJ software	ERP	CMS
Size	large	large	small
User group	private	professional	private and professional
ITS	LaunchPad	Jira	GITHub
Use of ITS	systematic	systematic	ad-hoc

B. Study Projects

This section provides characteristics of the three projects utilized for the study. Mixxx is a disk jockey software which implements basic features for managing and playing music and advanced features like a virtual mixer to perform seamless transitions between songs. Radiant is a content management system which implements basic features to create websites or blogs and advanced features like RSS feeds and an extension system to add 3rd party functionality. OFBiz is an enterprise automation software, where we studied the manufacturing resources planning component.

Table II provides further details on the projects. For OFBiz, only the manufacturing component and one corresponding provided feature was studied. The LOC of Mixxx comprise only the C++ code (excl. blanks and comments and XML configuration files). The LOC of Radiant comprise only Ruby and (r)html code (excl. blanks and comments). For Mixxx all blueprints and randomly sampled issues (to identify the quality of links between issues and blueprints) were analyzed, for Radiant all issues. In Radiant the status "implemented" was only identified for the feature-relevant issues.

TABLE II. Project Details

	Mixxx	**OFBiz**	**Radiant**
# features in list	22	1	10
Size (LOC)	94117	Not det.	33887
Programming language	C++ (& QT)	Java	Ruby (& Rails)
# issues	2211 + 113 blueprints + 138 user questions	120	348
# issues implemented	1239 + 59 blueprints	94	See text
# issues analyzed	50 + 113 blueprints	all	all
# analyzed issues with feature information	22 + 53 blueprints	19	50
# issues implemented and analyzed with feature information	22 + 53 blueprints	16	43
# subdivisions UD	14 chapters consisting of 69 sections	343	120 pages
# subdivisions UD analyzed	all	36	all
# subdivisions with feature information	62	34	64
# provided features identified in ITS	21	7	12
# provided features identified in UD	24	12	12

C. Study Operation

This section provides a short overview of the indicators we used for feature relevant information and describes how we searched the ITS and UD of the projects. We analyzed the data sources in February 2014. Moreover, we stored all analyzed data locally for a reliable reproduction of our results.

1) Feature indicators: For the ITS we looked for issues which describe a new functionality (F) or quality (Q). The feature has to be already implemented and the issue mentions F and Q or a component of F and Q. It was not always

 ISBN: 978-1-61208-367-4

easy to determine the implementation status of an issue. For Radiant the status was not managed explicitly. Thus, the implementation status was revealed by analyzing the comments of an issue and associated commits. For Mixxx and OFBiz we took the issues with the status "Implemented or Patch Available". We did not find any indication that the status for those issues was set wrongly. However, there might be issues which are implemented but the status is not set accordingly. For the UD we looked at section and page titles containing this kind of information and not only describing the operation of the product. The exact rules and corresponding examples are shown in Table III. We classified the feature information

TABLE III. INDICATORS OF FEATURE RELEVANT INFORMATION

ITS	UD
The issue is implemented AND mentions functionality or quality AND is not related to a bug AND is not only related to refactoring AND the term X or a component X_i of X is explicitly or implicitly mentioned.	*The item describes functionality or quality X and not operation (such as installing or getting help) AND the term X or a component of X is explicitly or implicitly mentioned.*
Radiant (Quality Performance, Component "Radius Parser" of "Radius Template Language"): **"Speed up Radius parser"**	Radiant Page Titles (Quality Performance and Caching) **"Disable caching in a radiant system"**
Radiant (Functionality Asset Management): **"Integrate an asset management solution"**	Radiant Page Title (Functionality Admin UI) **"Altering Tabs in the Admin UI"**
Radiant Implementation Status by comment: **"Seeing as there's a setting for this now, this issue can be closed?"**	Mixxx (Quality was not mentioned)
Mixxx (Quality User Experience, Functionality Vinyl Control): **"Improvements to the overall vinyl control user experience"**	Mixxx Section (Functionality Broadcast): **"Live Broadcasting Preferences"**
Mixxx (Functionality, Component Crates and Playlist): **"Currently Mixxx does not support hierarchies for crates and playlists. This, however, is possible"**	OFBiz (Functionality Routing Task): **"Find Routing Task"**
OFBiz: (Functionality Production Machines) **"cover the case in which many machines are used to complete a production task"**	

according to their abstraction levels. It is well-known that requirements and features are typically described on different abstraction levels. Based on the work of Gorschek et al. [7], we distinguish 3 levels of features:

- Requirements level (called feature level in [7]): the mentioned F comprises several functions or the Q affects several functions
- Function level: F or Q only refer to one function which a user can perform. Implementation details are not mentioned.
- Code level (similar to the component level used in [7], it focuses on the HOW): F or Q only refer to one function which a user can perform. Implementation details are mentioned. For UD the levels were easy to identify. Page or section titles referred generally to requirements, while subpages and subsections referred to functions. Code details were only mentioned in the UD of Radiant, as here the user is required to change classes to setup a certain functionality. Table IV shows examples for issue texts on different abstraction levels.

TABLE IV. EXAMPLES FOR ISSUE ABSTRACTION LEVELS

	Function	Quality
Requirements	**Radiant:** *"Break Radiant into several different extensions"* **OFBiz:** cf. Table III example bottom left.	**Radiant:** Internationalization
Function	**Radiant:** *"Errors when changing your password should be shown"* **Mixxx:** *"Implementation of a traktor library feature to allow professional DJs the smooth migration [...]."* **OFBiz:** *"Improve mrp to support to products which have no orders against them"*	**Radiant:** *"Make it so that pages are only cached for GETs"* **Mixxx:** *"Smooth Waveforms" (relates to a less stuttering display for track visualization).* **OFBiz:** *"There is a need to be able to block viewing info except that info that may pertain to that login"*
Code	**Radiant:** *"Javascript to stop you from navigating away from a page with changes"* **Mixxx:** *"It would be nice to be able to specify multiple <option>s for MIDI controls in XML mapping files."* **OFBiz:** *[...] accepts the partyId as a parameter, but has been commented [...] [however, the] functionality is vital for determining which employees are responsible for rejects*	**Radiant:** *"[Add] Ruby 1.9.x compatibility"* **Mixxx:** *"Distribute Launchpad translations with Mixxx Releases"*

In [9], Berry et al. distinguish typical section types of an UD: the abstractions (objects) of the domain (O) and the use cases (U). We use this distinction to classify the focus of a text. O is used when the feature is directly part of the UI or the software, while U is used when the feature requires some kind of dialogue to be used. U is not applicable to quality features. We also identified relationships between features. They are

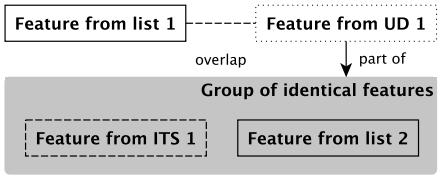


Fig. 1. Legend for the Feature Graphs.

used particularly in visualizations, such as Figure 2 and 4. Based on the information available to us, we determined the following relationships between the identified features (see Figure 1 for a legend of the relationships).

- Identical (features of different sources)

- Part of (features of different and same sources)
- Overlapping (features of different sources)

2) Procedure: The second and the third author conducted the actual search, while the first author acted as a reviewer. RQ1 was answered using thematic coding [12]. The second author coded the UD pages and sections and derived a set of codes characterizing the features described. Similarly, the third author coded issues in the ITS. The identified two feature sets were compared with the provided feature list on the project website and with each other. RQ2 was answered by comparing the different feature sets. The answer of RQ3 is based on the experience of the two authors during the manual derivation of the feature information.

D. Threats to Validity

We discuss the threats to validity according to Runeson et al. [13]: *Construct validity:* The authors have not been involved in the development of the sources. Thus, our view of what constitutes a feature of the software is clearly an external one, which might be different from what developers consider a feature of their software. To mitigate this threat, we used the feature list provided by the developers for comparison. *External validity:* The results are not representative for application software in general, as we only looked at three projects. For this exploratory study, we choose very different projects to enlarge the possible insights. *Reliability:* As only one researcher coded the ITS and the UD information, we cannot claim that other researchers would reproduce the coding. However, we used very explicit coding indicators and discussed them explicitly to minimize the bias of the individual coder. Moreover, the performed approach can be adapted to any software development project which provides the required data (ITS, UD and feature List).

III. Results

In the following, we answer the research questions for each project individually. The last subsection summarizes the insights for all projects.

A. Results of Project Radiant CMS

1) Provided feature list: The Radiant website contains a feature overview[2] which depicts 10 features using a name and a short one- or two-sentence description. As these features are listed prominently, we stipulate that they are the most marketing relevant for the developers. Table V shows these features and our classification as F or Q and O or U. The table also shows whether the feature was identified in the ITS or UD. Brackets indicate that the corresponding ITS or UD features are slightly different (see below).

[2]http://radiantcms.org/overview, accessed on August 8, 2014

TABLE V. Radianat Features

Provided feature list	Identified in
Built with Ruby on Rails (Q,O)	ITS
Custom Text Filters (F,U)	-
Flexible Site Structure (Q,O)	-
Intelligent Page Caching (Q,O)	ITS, UD
Layouts (F,O)	(UD)
Licensed under the MIT License (Q,O)	-
Pages (F,O)	ITS
Radius Template Language* (F,O)	ITS, UD
Simple Admin Interface (F,U)	ITS, UD
Snippets (F,O)	(ITS, UD)

* a special macro language (similar to HTML and Ruby).

2) Identification of feature information from UD and ITS (RQ1): The UD is organized in a wiki. The starting page of this wiki is a global table of contents. This table of contents is divided into 11 chapters, 8 of which only deal with administrative issues.

Thus, we identified the three chapters “The Basics”, “How Tos” and “Extensions” as primarily relevant for further analysis. “The Basics” contains seven links to top level UD pages. Except for the links to “FAQs” and “Getting Started”, the links point to pages describing Radiant features as mentioned in the feature list (Pages, Layouts, Snippets, Radius Tags, Customizing the Admin UI). In addition, there are six links to details of the Radius Tag feature and two links to details of the admin UI feature. Each top level UD page contains the intent and summary of the feature, screenshot of the features UI, and detailed descriptions of the feature use.

The “How Tos” chapter contains 29 links to top level UD pages. As visible by the titles, those links point to tutorials describing advanced features. The tutorials include usage examples and reference the basic feature pages. The Radius Template Language is referenced from almost all pages. The “Extensions” chapter starts with 6 pages describing the concept and usage of radiant extensions, followed by a list of 27 common extensions, and 11 pages which describe how to develop an extension for Radiant. According to the indicators of Table III, we identified 64 relevant pages.

The boxes marked with *UD* on the right side in Figure 2 shows the 13 features identified from the UD. *Content delivery* refers to different channels like RSS, *content location* to search in a web page. Most pages are on the function level and many describe layout. The number of pages related to a feature do not signify the importance of that feature. The features listed under “The Basics” can be seen as most essential, however, they are described on 15 pages, only.

The Radiant project uses GitHub as ITS. It is used for different aspects, such as Feature Requests, Bug Reports, Discussions of the development process, Discussions about refactorings and sometimes User Problems and Discussions about Documentation.

GitHub provides optional labels to classify an issue. Since the labels are optional, they are rarely used in the Radiant project. This implies that issues related to features, bugs, or

 ISBN: 978-1-61208-367-4

other aspects of Software Engineering (SWE) are not labeled accordingly by any means.

Therefore, we analyzed each of the 348 issues manually and derived their category (feature, bug, refactoring, other SWE aspects) by analyzing the descriptions and comments.

The boxes marked with *ITS* on the left side in Figure 2 refer to the 11 features identified from the ITS. *Asset management* refers to content different from the pages such as image files. *Development* comprises support for developers, such as a framework, *Frontend* refers to usability features. The issues mostly deal with individual functions, half of them deal with code (cf. Figure 3a). Many issues deal with the *Simple Admin UI*. Here again, the number of issues does not signify the importance of the feature. Furthermore, there are issues with many comments, but, e.g., a very short implementation.

3) Commonalities and differences of UD and ITS and provided features (RQ2): Figure 2 shows the relationships between the identified features. As could be expected, the description of the features in the ITS is quite often on the code level, while the UD features are described on all three levels. Almost a third is on the code level which is unusual for an UD. This is due to the fact that code needs to be changed for some functionalities. However, only 2 features are solely described on the code level. Figure 3 shows that the feature sets have

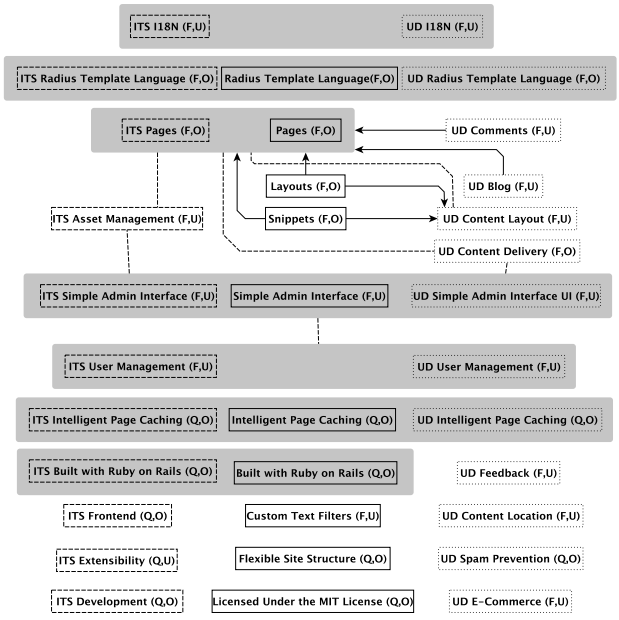
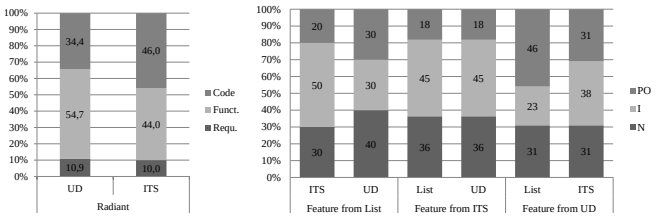


Fig. 2. Radiant Feature Graph (transitive relationships are not shown)

some commonalities, but also differences. Almost half of the ITS features (45%) are identical to the provided features, while only a third of the UD features (31%) is identical (cf. Figure 3b). This might be due to coder differences, but also due to the fact that the UD already provides a structure indicating low-level features which are not mentioned on the marketing level. Issues mention the features without any structure. Thus, the developer and the coder are missing a structure when referring to low-level features. Provided features not identified



(a) Abstraction Level (b) Feature Relationships (N = Not Related, I = Identical, PO = Partial Or Overlapping)

Fig. 3. Radiant Commonalities and Differences of UD, ITS and Feature List

from the ITS (18%) may be due to the fact that the ITS was not used from the beginning of the development. The basic functionality of the software was implemented before the ITS was used. For ITS features not in the provided list (31%), the content could be a reason. While Internalization and Extensibility seem relevant as prominent features, Fronted and Development issues might be too low-level. It is interesting to note that all ITS features which do not have a relation to either the list or the UD are quality-related. All features identified from the UD seem relevant, although 31% of them are not in the provided list. There is no pattern wrt F/Q or O/U in the differences between UD and the other feature sets. The PO-relationships between Pages, Layouts and more fine-grained features, such as Blog or Comments, show that granularity is a challenge. Only UD features have Part-of-relationships. Again, this can be due to the more fine-grained structure of the UD. UD features are more closely related to ITS feature (45% identical) than the provided list, but there are almost as many (36%) non-related features.

4) Automatic identification of feature information (RQ3): Most feature-related information was identified in older issues. 34 feature-related information items could be found in issues #1 to #68. 16 feature information items in #71 to #202 and no feature-related information was found in #203-#384. This suggests that a) older issues should be available for automatic feature extraction and b) it might be best if the ITS is used from the beginning of the development (e.g., design and prototyping phases). In Radiant however, the ITS was only used after a prototype of the software had already been built.

For the ITS we searched for text containing '*should*be*', '*add*, '*would*be*' and '*allow*user*' in the issue title and description. This revealed 27 of the 43 issues with feature related information. However, the search added about the same amount of noise and included refactoring- and bug-related issues. Therefore, the precision of this approach is relatively low. However, depending on the usage of an ITS, it might be possible to extract more precise search terms. A pitfall in automatic analysis are the tags of the Radiant ITS. As in [10], manual categorizations are often wrong. Although tags like *bug*, *design*, or *javascript* are introduced in Radiant, they are not used consistently. Since tags are optional, we found that most issues are not tagged at all. The *bug* tag is only used for

 ISBN: 978-1-61208-367-4

one issue, which is not reliable for any automatic extraction. Further analysis, for example topic analysis [14], is necessary to identify feature labels.

For the automatic identification of features from the UD, the relevant pages need to be identified in a first step. For Radiant the relevant pages were mainly contained in three chapters, which can be identified more efficiently manually than automatically. Additionally, some pages are only related to system operation and do not contain feature information. To some extent, these pages could be identified by searching for operation-related terms such as installation in order to discard these pages. Another input for the identification of relevant pages is the linkage structure. Based on this, it is possible to identify frequently referenced pages which most likely are feature-relevant pages.

The identification of the feature labels could start from the page titles. The nouns contained in the titles can be used as a starting point to create a feature list.

B. Results of Project Mixxx

TABLE VI. RADIANT FEATURES

Provided features	Identified in
Advanced Controls (F,U), Dual Decks (F,O)	IST,UD
Decks: Beat Looping. Broad Format Support, Hotcues, Intuitive Pitchbends, EQ and Crossfader Control, Time Stretch and Vinyl Emulation (F,O)	(ITS, UD)
Designer Skins (F,O)	ITS
Free Timecode Vinyl Control (F,U)	ITS,UD
Microphone Input (F,U)	ITS,UD
MIDI Controller Support (F,U)	(ITS,UD)
Powerful Library: Auto DJ, BPM Detection and Sync, Crates and Playlists, Disk Browsing, iTunes Integration (F,U)	ITS,UD
Quad Sampler Decks (F,O)	UD
Recording (F,U)	ITS,UD
ReplayGain Normalization (F,U)	-
Shoutcast Broadcasting (F,U)	(ITS,UD)

1) Provided feature list: Similarly to Radiant, the list of provided features was taken from a website [3] with short marketing descriptions. The feature list contained 20 functional features (see Table VI). Features which are part of a more general feature (e.g. library or deck) are listed in one row.

2) Identification of feature information from UD and ITS (RQ1): The UD is part of a general documentation WIKI which also contains developer documentation and documentation for special users, like artists. We focused on the user manual. The manual contains 14 chapters, 9 of which are feature relevant. These 9 chapters contain 69 sections. Since the chapters contain an introductive text, we assigned feature labels (abstraction level requirement) to 8 of them (one chapter title was "advanced features") and to the 54 sections which satisfied our indicators of TABLE III (abstraction level function). None were on the code level. The boxes marked with *UD* on the right side in Figure 4 show 20 of the

[3]http://mixxx.org, accessed on August 8, 2014

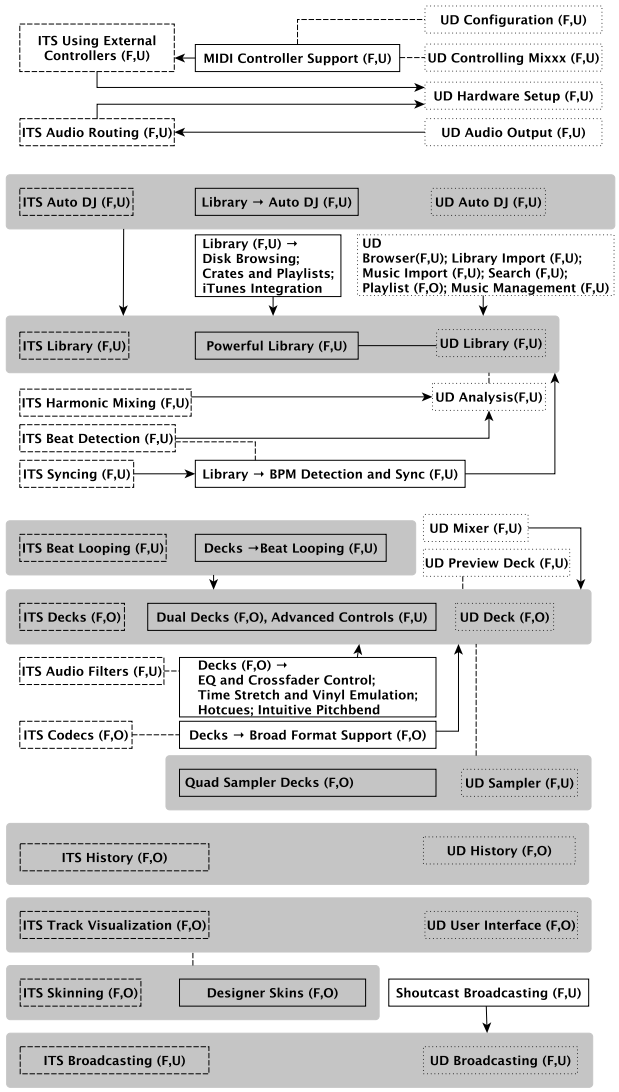


Fig. 4. Mixxx Feature Graph (identical, not related features, and transitive relationships are not shown)

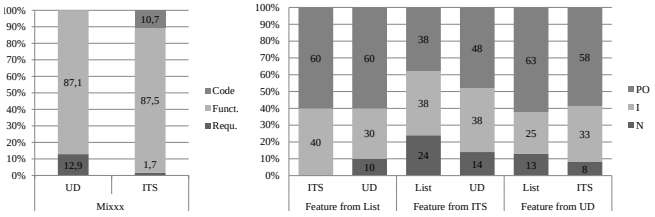
24 identified features and their classification (the features *DJing(F,U), Microphone (F,U), Recording (F,U)* and *Vinyl Control (F,U)* are not linked to the provided features and thus have been omitted in the Figure). The feature *Analysis* refers to the preparation of harmonic mixing, *Controlling Mixxx* allows setting device specific options, and *Vinyl control* allows to use records to control digital playback. All of the identified features describe a functionality. The number of sections or chapters corresponds to the complexity of the features. *Music Management* is the only feature described in detail (7 pages) which is not related directly to a chapter.

The Mixxx Project uses Launchpad as ITS. It contains 2211 issues (including bugs and feature requests), 113 so-called blueprints and 138 questions. The issue classification in bugs and features as made by the developers is very reliable for the issues we analyzed. The blueprints describe refactorings and higher level requirements for features. Blueprints and issues

 ISBN: 978-1-61208-367-4

are often linked, and issues are often linked with the code, but not always. Since blueprints contain more feature-relevant information than issues in the way Launchpad is used in the project, we analyzed all 113 blueprints for feature information. In our analysis, we included 59 blueprints with the status implemented.

The boxes marked with *ITS* on the left side in Figure 4 refer to 15 of the 21 features identified from the ITS (*Development (Q,O), Internationalization (F,O), Microphone Usage (F,U), Playback (F,U), Recording (F,U)* and emphVinyl Control (F,U) are not linked to the provided features and thus have been omitted in the Figure). All of the identified features describe functionality. *Beat Detection* analyzes the speed of a track. *Beat looping* repeats a short part of the track. *Codecs* are different digital formats. As for Radiant, *Development* describes support for the developers. *Skinning* refers to different UI looks. *Syncing* matches the speed of different songs for the mix. Only few blueprints are on the code level.



(a) Abstraction Level (b) Feature Relationships (N = Not Related, I = Identical, PO = Partial or Overlapping)

Fig. 5. Mixxx Commonalities and Differences of UD, ITS, and Feature List

3) Commonalities and differences of UD and ITS and provided features (RQ2): Figure 4 and 5 illustrate the commonalities and differences of the UD, ITS, and provided feature sets. The graph in Figure 4 does not show the identical features AutoDJ, Microphone Input, Recording and Vinyl Control, as well as the 3 features of ITS (Development, Internationalization, Playback) and one feature each of UD (DJing) and List (Replay) which are not related to other features.

There are fewer provided features (a tenth) compared to Radiant (roughly a third) which have not been mentioned by UD or ITS. This was expected from the fact that the ITS blueprints and the UD seem to be well maintained. Similar to Radiant, more features from the provided list were directly identified from the ITS (40%) compared to the 30% of the provided features identified from the UD. However, the difference is smaller. Furthermore, there are more ITS features (24%) which are not related to provided features. Two of them are related UD features. There are many identical features between ITS and UD (between 30 and 40%) and very few non-related ones($< 10\%$). However, there are also many part-of-relationships between ITS features and either UD (48%) or the provided features (38%). And there are even more PO-relationships from UD features to either the ITS (58%) or the provided features (63%). This indicates that even when well-maintained, the granularity in the different sources is different. The distinction between F und Q is not relevant as no Q features were involved. Again, no pattern wrt O/U could be found.

4) Automatic identification of feature information (RQ3): As the Mixxx ITS is maintained very systematically, the possibilities to categorize the status (e.g. implemented, draft, in progress) as well as the issue (e.g. bug, wishlist, blueprint) could be used as indicators for feature relevant information. The identification of the feature labels remains a problem.

The Mixxx UD is a well-structured document separated into chapters and two section levels. 9 of the 14 chapters are feature-relevant, compared to 3 out of 11 for Radiant. Often the chapter and section titles directly contain feature-relevant terms. As for Radiant the relevant chapters can be identified just by manually looking at the chapter titles. For the identification of features on the requirement abstraction level, the chapter titles can be used. The section titles on the first section level can be used for feature identification on the function abstraction level.

C. Results of Project Apache OFBiz

1) Provided feature list: The project OFBiz was studied only partially. As the project is very large a complete analysis was not feasible. The feature page[4] lists features on the requirements level. We decided to look at the manufacturing feature (one component) and the corresponding UD and issues, only.

2) Identification of feature information from UD and ITS(RQ1): The UD for OFBiz is organized as a wiki. However, the wiki only contains more or less empty pages and some basic structures (e.g. sections for role specific documentation, e.g., for managers). Based on this fact, we decided to use the outdated Manager Reference for our UD analysis (last updated in 2004, uploaded to the wiki as PDF attachment between 2006-12 and 2007-01). Based on the experience gained from the previously analyzed projects, we looked at the chapter and section headings, only. The document contains

TABLE VII. OFBiz Manufacturing Features from UD and ITS

Feature UD	Feature ITS
Bill of Materials (F,U)	Data Security (Q,O)
Bill of Mat. Simulation (F,U)	Internationalization (F,O)
Calendar (F,U)	Manage orders (F,U)
Job Shop (F,U)	Manage Products (F,U)
Manufact. Res. Planning (F,U)	Manage Production Machines (F,O)
Manufacturing Rules (F,U)	Manage Production Runs ()
Production Run (F,U)	Resource Planning (F,U)
Reports (F,O)	
Requirement Verification (F,U)	
Routing (F,U)	
Routing Task (F,U)	
Shipment Plans (F,U)	
Status Report (F,U)	

[4]http://OFBiz.apache.org, accessed on August 8, 2014

 ISBN: 978-1-61208-367-4

343 subdivisions organized into 4 hierarchy levels. We decide to remove the sections on the 3rd and 4th hierarchy level, since they only refer to single attributes, e.g. of specific input form values. 8 chapters and 28 sections remained. From this we could identify 13 distinct features (cf. Table VII, left column). 8 of the features are on the requirement abstraction level, based on the chapters. The other 5 features are mostly additional aspects of the requirement abstraction level features, i.e. they have been identified in subsections of the respective chapters. As expected for a manager reference, code details were not mentioned. Also, quality features were not mentioned and only one object of the domain.

The OFBiz Project uses Jira as ITS. It contains 5567 issues, 120 of these issues are related to the manufacturing component which was determined by filtering the ITS. We analyzed all 120 issues of the manufacturing component and identified 7 features (cf. Table VII right column). Security was the only quality aspect identified.

In OFBiz the issue feature descriptions are generally longer (in terms of words) as in the other projects. Requirements, for example are described in detail and often include multiple solution ideas (though not mentioning concrete code), e.g. **"[...] this can be implemented in many ways: a) expanding the concept of Fixed Asset groups [...] b) (more complex) add new association entities to link a task [...]"**. Although this suggests a very accurate handling of the ITS, we found multiple misclassifications of issues (e.g. bugs classified as improvements). In addition, some features were distributed over many issues. E.g. I18N included multiple issues for every single language and one main issue describing the feature and none of these were linked.

TABLE VIII. COMMONALITIES AND DIFFERENCES OF ITS AND UD

Feature ITS (7)	Feature UD (13)	Map
Data Security (Q,O), Internationalization (F,O)		
Manage Orders (F,U)	Bill of Materials (F,U), Bill of Materials Simulation (F,U), Calendar (F,U). Shipment Plans (F,U)	(O)
Manage Products (F,U), Manage Production Machines (F,O)	Manufacturing Rules (F,U)	(O)
Manage Production Runs (F,U)	Production Run (F,U)	I
Ressource Planning (F,U)	Manufacturing Resource Planing (F,U) Job Shop (F,U), Reports (F,O), Requ. Verification (F,U), Routing (F,U), Routing Task (F,U) Status Report (F,U)	I

3) Commonalities and differences of UD and ITS and provided features (RQ2): As the descriptions of the UD were coarse and we mainly looked at the headings, we did not derive a full mapping. Table VIII shows a rough mapping of the features of UD and ITS. Two features are identical and a few have overlaps. However, almost half of the UD features were not mentioned in the issues. This can be explained by the fact that the UD was outdated.

4) Automatic identification of feature information (RQ3): The OFBiz analysis did not reveal any new insights wrt automatic identification. For the UD we only used the chapter and section titles manually. Thus, this could be also the basis for an automatic identification. As for Radiant, most feature related information was identified in older issues. In the ITS, we found a feature ratio over all issues from 10 to 18% between 2006 and 2009 and only about 3 to 5% between 2009 and 2013.

D. Overall Results

1) Identification of feature information from UD and ITS (RQ1): ITS as well as UD can serve to identify features. The features, however, are described on different abstraction levels [7] (cf. Figure 6). For both, Mixxx, and OFBiz the UD does not contain features on code level and $> 75\%$ on function level. In contrast, the UD of Radiant mainly contains feature information on the code and function levels. In the ITS, feature information is found on all three levels, but, similar to the UD, there are only few features on the requirements level, and the distribution of abstraction levels in the ITS is quite different for each of the projects. Quality features are typically not

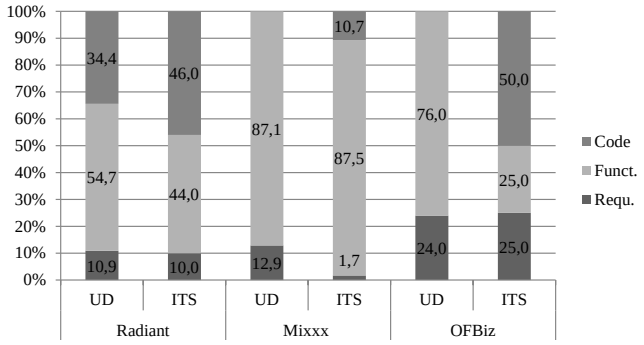


Fig. 6. Abstraction Levels

mentioned in the UD. Radiant and OFBiz UD's mention few quality features and Mixxx's UD none. Most quality features were found in the ITS.

2) Commonalities and differences of UD and ITS and provided features (RQ2): There is a noticeable overlap between the feature information in the ITS, the UD, and the provided feature list. In the Radiant project, roughly a third of the listed features could not be identified by UD or ITS, in the Mixxx project only a tenth could not be identified. Similarly, for Radiant only a third and for Mixxx only a tenth of the features was not related between ITS and UD. This indicates that, both ITS and UD could be used to record feature information systematically.

As the ITS is mainly important for the developers and the UD is targeted to the users, it seemed more likely that the UD better records the listed features [9]. However, it turned out that UD and ITS record the listed features equally well. It is interesting to note that in both projects 30-50% of the

 ISBN: 978-1-61208-367-4

features were identical (between List and UD, List and UD, and ITS and UD). This means that in the case of Mixxx there is a high percentage of overlapping features, while in the case of Radiant there are few overlapping features. Thus, even for a systematically documented project like Mixxx, a feature representation generated from UD or ITS would be different from the marketing feature list. Thus, different feature representations are likely needed for different purposes.

3) Automatic identification of feature information (RQ3): We have preliminary insights for automatic identification. It seems feasible to manually delimit the relevant pages of the UD and to focus on page or section titles to identify feature labels. Also for the ITS, there are first ideas to delimit the relevant issues, but the label identification will require more sophisticated techniques. As neither ITS nor UD were a perfect source, it seems likely that at least both sources must be searched and combined to yield a complete feature set.

Although best practices in RE suggest to describe new features as as-is and to-be situations, we found only one issue which mentions both situations. Generally, the to-be situation was described and the as-is situation was implicit. Furthermore, the quality and use of language, the quality of descriptions as well as categories and links differ from project to project. An automatic identification therefore needs to be "tuned" for each project.

From our experience, we see the following hints for using ITS and UD to record features instead of setting-up a separate feature documentation:

- The Mixxx project shows that feature information can be explicitly managed within an ITS, if it is separated from (but linked to) the usual stream of bugs and change requests. Furthermore, it seems likely that issues in ITS could profit from an abstraction classification or traces to more abstract information.
- Berry et al. recommend structuring an UD into objects, use cases and advanced features [9]. The UD of the projects have some similarities to this structure, but this could be improved. It might be helpful to structure the feature list accordingly.
- Blueprints and issues are much simpler to allocate to software components, because both have a technical nature. Without detailed knowledge of the software architecture, this is almost impossible for UD. Thus, if relationships between features and software components are important, ITS should be used as a source.

IV. Related Work

In this section, we discuss related work which derives feature information from diverse sources manually or semi-automatically.

Ghazarian identified generic classes of software functionality from 15 different requirement specifications in the domain of web-based enterprise systems. The identified classes such as data input or user interface navigation could potentially be useful as indicators of feature information [15]. They also describe that much feature-related information could be found and categorized analyzing only a small amount of issues, respectively only section and paragraph names in the UD. Their classification, however, is very technical and uses low abstraction levels. In contrast, our work classified features for different abstraction levels.

Noll et al. analyze an open source project to identify by whom and where requirements are proposed [16]. They select 13 given features and then trace them. In contrast we first looked at our data sources to manually identify features and then compared them with the given feature list. Thus, we gathered more data about how features are described in detail.

The following approaches use text mining to derive feature-like information from requirements specifications. Thus, their data sources are much more elaborate than the feature descriptions in ITS. Although the quality of UD and of the requirements specifications could be comparable, requirements specifications are rather structured than UD. Thus, results from both document sources are not quite comparable. These approaches can be used as a starting point for our future work on a semi-automatic feature-derivation approach. Gacitua et al. provide an approach for identifying abstractions from text documents which outperforms the usual information-retrieval methods [17]. In [18], Boutkova and Houdek describe an approach applied in industry to derive features from requirements documents based on a list of nouns. In the study it provided helpful input for experts.

Kuhn et al. recover topics from source code [19] and note that some extracted clusters represent software features. Since the features are extracted from source code, they are on the functional level (for example *handling text buffers*). In contrast, this paper is interested in feature descriptions on different abstraction levels.

V. Conclusion and Lessons Learned

The exploratory study of the OSS projects has shown commonalities and differences of feature information in UD and ITS and provided feature lists. The results are promising in the sense that ITS and UD provided relevant features. The results also show that deriving a complete feature set semi-automatically will be very difficult and will depend on the project.

In the projects we studied, the feature information within the project was consistent and some feature descriptions formed patterns (e.g. headings in the UD often denoted features). However, most of these patterns were not transferable to other projects.

During our research, we found that feature information is contained in only few issues of an ITS. Due to an ITS's nature, other issues like bugs are also tracked. Although many ITS provide the option to categorize issues manually as feature or bug, the quality of those manual categories depends largely on the project. Therefore, an analysis of the natural language is

 ISBN: 978-1-61208-367-4

also needed to identify all feature information. Furthermore, the feature information can be scattered all over the issue and can be found in title, description or even comments (although title and description are most common). Therefore, only a very small part of the natural language in an issue contains the feature description apart from rationale, solution ideas, social interaction and so on.

For UD, the starting point to detect and extract feature data semi-automatically is the structure of the respective documents. The analysis of the projects in this study showed that different structure levels in UD map to different feature abstraction levels. Our first experiments in the direction of using the UD headings and applying filter operation to remove common conjunctions seems promising. Moreover, certain UD parts, like administrative instructions, can be omitted for feature derivation since they do not contain feature-relevant data. To create a feature list from ITS and UD data automatically, a structured way of storing feature information would be necessary.

In future work, we will develop semi-automatic feature identification algorithms as well as guidelines for maintaining feature information in UD and ITS. They will be applied in industry to explore whether the identified features can be successfully used in release management.

ACKNOWLEDGMENT

This work is partly funded by the Bonn-Rhein-Sieg University Graduate Institute. We thank the Open Source community for the freely available data that was used in this research.

REFERENCES

[1] S. Fricker and S. Schumacher, "Release planning with feature trees: industrial case," in *Requirements Engineering: Foundation for Software Quality. 18th International Working Conference, REFSQ 2012*, vol. LNCS 7195. Essen, Germany: Springer Berlin Heidelberg, 2012, pp. 288–305.

[2] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feasibility Study Feature-Oriented Domain Analysis (FODA) Feasibility Study," Software Engineering Institute, Carnegie Mellon University, Tech. Rep. November, 1990.

[3] P. Shaker, J. M. Atlee, and S. Wang, "A feature-oriented requirements modelling language," in *2012 20th IEEE International Requirements Engineering Conference (RE)*. IEEE, Sep. 2012, pp. 151–160.

[4] T. a. Alspaugh and W. Scacchi, "Ongoing software development without classical requirements," in *2013 21st IEEE International Requirements Engineering Conference (RE)*. Rio de Janeiro, Brazil: Ieee, Jul. 2013, pp. 165–174.

[5] B. Paech, R. Heinrich, G. Zorn-Pauli, A. Jung, and S. Tadjiky, "Answering a Request for Proposal Challenges and Proposed Solutions," in *18th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 12)*, vol. LNCS 7195. Essen, Germany: Springer, 2012, pp. 16–29.

[6] G. Zorn-Pauli, B. Paech, T. Beck, and H. Karey, "Analyzing An Industrial Strategic Release Planning Process A Case Study At Roche Diagnostics," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*, vol. LNCS 7830. Springer, 2013, pp. 269–284.

[7] T. Gorschek and C. Wohlin, "Requirements Abstraction Model," *Requirements Engineering Journal*, vol. 11, no. 1, pp. 79–101, Nov. 2006.

[8] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code. A taxonomy and survey," *Journal of Software: Evolution and Process*, vol. 25, no. 1, pp. 53–95, 2013.

[9] D. M. Berry, K. Daudjee, I. Fainchtein, J. Dong, M. A. Nelson, T. Nelson, and L. Ou, "User s Manual as a Requirements Specification : Case Studies," *Requirements Engineering*, vol. 9, no. 1, pp. 67–82, 2004.

[10] K. Herzig, S. Just, and A. Zeller, "Its Not a Bug, Its a Feature: How Misclassification Impacts Bug Prediction," in *Proceedings of the 2013 International Conference on Software Engineering (ISCE)*. IEEE Press, 2013, pp. 392–401.

[11] R. K. Yin, *Case Study Research: Design and Methods*, 5th ed., ser. Applied Social Research Methods. SAGE Publications, Inc., 2013.

[12] C. Robson, *Real World Research*, 3rd ed. Wiley, 2011.

[13] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, Dec. 2009.

[14] D. M. Blei, "Probabilistic topic models," *Communications of the ACM*, vol. 55, no. 4, p. 77, Apr. 2012.

[15] A. Ghazarian, "Characterization of functional software requirements space: The law of requirements taxonomic growth," in *2012 20th IEEE International Requirements Engineering Conference (RE)*. Chicago, Illinois, USA: IEEE, Sep. 2012, pp. 241–250.

[16] J. Noll and W.-M. Liu, "Requirements elicitation in open source software development: a case study," in *Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development - FLOSS '10*. New York, New York, USA: ACM Press, 2010, pp. 35–40.

[17] R. Gacitua, P. Sawyer, and V. Gervasi, "Relevance-based abstraction identification: technique and evaluation," *Requirements Engineering*, vol. 16, no. 3, pp. 251–265, Jun. 2011.

[18] E. Boutkova and F. Houdek, "Semi-automatic identification of features in requirement specifications," in *2011 IEEE 19th International Requirements Engineering Conference*. Trento, Italy: Ieee, Aug. 2011, pp. 313–318.

[19] A. Kuhn, S. Ducasse, and T. Gîrba, "Semantic clustering: Identifying topics in source code," *Information and Software Technology*, vol. 49, no. 3, pp. 230–243, Mar. 2007.

 ISBN: 978-1-61208-367-4

Towards Duplication-Free Feature Models when Evolving Software Product Lines

Amal Khtira, Anissa Benlarabi, Bouchra El Asri
IMS Team, SIME Laboratory, ENSIAS, Mohammed V Souissi University
Rabat, Morocco
amalkhtira@gmail.com, a.benlarabi@gmail.com, elasri@ensias.ma

***Abstract*—Since the emergence of Software Product Line Engineering, the requirements evolution issue has been addressed by many researchers and many approaches have been proposed. However, most studies focused on evolution in domain engineering while application engineering has not received the same attention. During the evolution of a derived product, new features are added or modified in the application model, which may cause many model defects, such as inconsistency and duplication, both in application model and between the latter and the domain model. The aim of this paper is to propose a framework that enables to avoid duplication when evolving software product lines.**

Keywords*—*Software Product Line; Requirements Evolution; Domain Engineering; Application Engineering; Duplication.

I. INTRODUCTION

The Software Product Line Engineering (SPLE) has emerged as a paradigm whose main objective is to develop software applications based on a core platform. The adoption of this approach by companies enables them to reduce time to market, to reduce cost and to produce high quality applications. Another major advantage of the PLE is the reuse of core assets to generate specific applications according to the need of customers.

The SPLE approach consists of two processes, namely, domain engineering and application engineering [1]. During these processes, a number of artefacts are produced which encompass requirements, architecture, components and tests. Domain engineering involves identifying the common and distinct features of all the product line members, creating the design of the system and implementing the reusable components. During application engineering, individual products are derived based on the artefacts of the first process, using some techniques of derivation.

Many issues related to SPLE have been addressed both by researchers and practitioners, such as reusability, product derivation, variability management, etc. The focus of our study will be on SPL evolution. Evolution is defined by Madhavji et al. [2] as "*a process of progressive change and cyclic adaptation over time in terms of the attributes, behavioral properties and relational configuration of some material, abstract, natural or artificial entity or system*". This definition applies to different domains, including software engineering.

In the literature, several studies have dealt with evolution in Software Product Lines (SPLs). Xue et al. [3] presented a method to detect changes that occurred to product features in a family of product variants. In order to support agile SPL evolution, Urli et al. [4] introduces the Composite Feature Model (CFM), which consists of creating small Feature Models (FMs) that corresponds each to a precise domain. Other approaches, such Ahmad et al.'s [5], focused on the extraction of architecture knowledge in order to assess the evolutionary capabilities of a system and to estimate the cost of evolution. Some papers focused on the co-evolution of different elements of SPLs [6].

Based on the literature, we have found that most of the studies addressing software evolution focus on domain engineering, while application engineering has not received the same interest. However, the experience has proven in many industrial contexts that systems continue to change even after the product derivation. This change can be the source of many problems in the product line such as inconsistency and duplication. Indeed, the core assets of the product line and the artefacts of derived products are most of the time maintained by different teams. Moreover, developers under time pressure can forget to refer to the domain model before starting to implement the changes. For these reasons and others, duplication in SPL can easily happen. We consider Duplication the fact of adding to the application model features of the same semantics, which means that they satisfy the same functionality. In this paper, we propose a framework that deals specifically with the problem of duplication when evolving products in application engineering.

The remainder of the paper is structured as follows. Section 2 gives an overview of the background of our study and describes the problem we are dealing with. In Section 3, we present the basic concepts and the overview of the proposed framework. In Section 4, we provide a formalization of the basic concepts before describing the algorithm of deduplication. An application of the framework on a case study is presented in Section 5. Section 6 positions our approach with related works. The paper is concluded in Section 7.

II. BACKGROUND

In this section, we introduce the background of our study. First, we present the SPLE paradigm, then we give an insight on the problem of duplication when evolving products in application engineering.

A. Software Product Line Engineering

A SPL is defined by Clements and Northop [7] as "*a set of intensive-software systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way*". The main goals of a SPL are to reduce the cost of developing software products, to enhance quality and to promote reusability.

The domain engineering phase of the SPLE framework is responsible for defining the commonality and variability of the applications of the product line. Capturing the common

 ISBN: 978-1-61208-367-4

features of all the applications increases the reusability of the system, and determining the variant features allows the production of a large number of specific applications that satisfy different needs of customers. In order to document and model variability, many approaches have been proposed. Some of them proposed to integrate the variability in the existing models, such as UML models or feature models (FORM [8]). Pohl et al. [1] preferred to define it separately in a dedicated model, i.e., the orthogonal variability model. Another approach proposed by Salinesi et al. [9] used a constraint-based product line language. When the model is ready, the next step consists of creating the design of the system which contains the software components and their relationships. Those components are then implemented and the code of the product line is generated.

The process of creating a specific product based on a SPL is referred to as product derivation or product instantiation. Product derivation consists of taking a snapshot of the product line by binding variability already defined in the domain engineering and using it as a starting point to develop an individual product. This process is applied during application engineering phase and is responsible for instantiating all the artefacts of the product line, i.e., model, design, components, etc.

B. Duplication of Features during SPL Evolution

The goal of SPLE is to make an up-front investment to create the platform. Indeed, during domain engineering, the requirements of all the potential applications are captured, and as far as possible, the scenarios of the possible changes have to be predicted and anticipated. The evolution and maintenance of the product line are conducted through several iterations until the platform becomes as stable as possible. As new evolutions arise, the domain artefacts are adapted and refined.

On the one hand, the team responsible for developing and maintaining the product line studies the requirements of each customer and derives specific applications that respond to these requirements. On the other hand, a different team takes in charge the maintenance of each application. Following the logic of SPLE, the derived applications are not supposed to change much, but the experience has shown that this assumption is not always true. In fact, even after the derivation of a specific product, new demands can be received from the customer, either changes to existing features or addition of new ones.

During the maintenance of a product, duplication of knowledge can easily happen when evolving the model, the design or the code. In [10], four categories of duplication are distinguished:

- Imposed duplication: Developers cannot avoid duplication because the technology or the environment seems to impose it.
- Inadvertent duplication: This type of duplication comes about as a result of mistakes in the design. In this case, the developers are not aware of the duplication.
- Impatient duplication: When the time is pressing and deadlines are looming, developers get impatient and tend to take shortcuts by implementing as quick as possible the requirements of customers. In these conditions, duplication is very likely to happen.
- Inter-developer duplication: Different people working on one product can easily duplicate information.

In the context of SPLE, at least the three last categories might occur. Indeed, when a derived application is shipped, developers responsible for maintaining it do not have a clear visibility of the domain model because another team conceived it. Thus, developers of the application may add features which are already satisfied in the domain model and have only to be derived or configured. In addition, under time pressure, developers do not refer to the application model and might add features which are already implemented. To the best of our knowledge, few attempts have dealt with duplication in the application engineering. The aim and contribution of this paper is to provide a framework that helps developers avoid duplication in a SPL when evolving a specific product.

III. A Framework to Avoid Duplication when Evolving Derived Products

In this section, we first provide a short definition of the basic concepts used in the framework, then we present the overview of the framework.

A. Basic Concepts

Before going any further, we will give an insight of the basic concepts used in the framework.

Domain Model: A domain is a family of related products, and the domain model is the representation of all the different and common features of these products. There are many types of domain models, but the most interesting are the feature model [8] and the variability model [1].

Application Model: The model corresponding to an individual application. It is generated by binding the variability of the domain model in a way that satisfies the needs of a specific customer [1].

Feature: A feature is the abstraction of functional or non-functional requirements that help characterize the system and must be implemented, tested, delivered, and maintained [8][11]. A feature is either:

- *Mandatory*: it exists in all products.
- *Optional*: it is not present in all products.
- *Alternative* (One Of): it specializes more general feature; only one option can be chosen from a set of features.
- *Or*: One or more features may be included in the product.

Variation Point: Variation points are places in a design or implementation that identify the locations at which variation occurs [12].

Variant: It is a single option of a variation point and is related to the latter using a variability dependency [13].

 ISBN: 978-1-61208-367-4

Specification: Requirements specification is a description of the intended behavior of a software product. It contains the details of all the features that have to be implemented during an evolution of the system.

Specific Variant or Variation point: We consider a variant or a variation point as specific when they concern a particular need of an application that belongs to the product line (e.g., features related to confidential data, features that need legal authorization).

Generic Variant or Variation point: We consider a variant or a variation point as generic if they can be demanded by many applications of the product line (e.g., ergonomic or utility features, non-functional features).

B. The Framework in a Nutshell

With the large number of features in the SPLs, the manual checking of duplication becomes a complicated and an error-prone task. In order to deal efficiently with the problem of duplication during the evolution of derived products, we propose the framework depicted in Figure 1 as an attempt to set an automated deduplication tool.

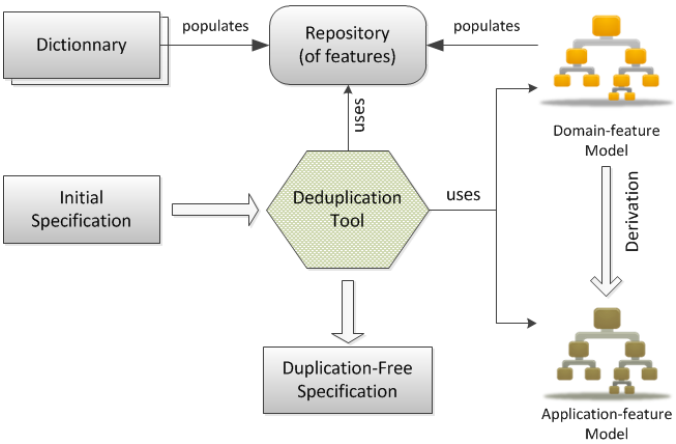


Figure 1. The overview of the framework.

Initial Specification: In this framework, we take as an input the specification of a new evolution related to a derived product. This specification contains the requirements that have to be implemented in this specific product. To use these requirements, we need to express them as features using the FODA feature model [14]. In the context of our framework, we consider that a feature is the association of a variation point and a variant. To create the feature model, we opt for FeatureIDE [15]. This tool enables to graphically create the feature model and the associated XML is generated automatically.

Domain and Application Models: The main prerequisites of the framework are the domain model and the application model. To create these models, we use the FeatureIDE tool in order to generate the sources in the form of XML files.

Repository: The repository contains the features of the domain model and also the set of all the possible synonyms and alternatives for the concepts used in the product line. The elements of the repository are defined using the Resource Description Framework (RDF). RDF [16] is a W3C recommendation that supports semantic interoperability between different resources on the web.

Deduplication Tool: This tool contains a set of algorithms of features verification. In this paper, we focus on the algorithm of deduplication. Before describing the algorithm, we need to define some predicates.

Equivalence: We consider that a variation point (resp. a variant) is equivalent to another variation point (resp. variant) if they both implement the same functionality, which means that they have the same semantics. We define the function *Equiv* which can take three values:

$$Equiv(x) = x_0 \Rightarrow \begin{cases} x_0 = x \\ x_0 \text{ is equivalent to } x \text{ and } x_0 \neq x \\ x_0 \in \emptyset \end{cases}$$

Example: The variant "On-line Sales" associated to the variation point "Sales" is equivalent to the new variant "e-sales" (cf. Section 5).

Duplication: We consider that a feature of the specification is duplicated if the associated variation point and variant have equivalents in the application model or the domain model.

The aim of the algorithm is thus to verify the non-duplication of all the features of the initial specification in order to generate a new correct specification. Indeed, for each feature of the initial specification, the algorithm verifies whether the associated variation point and variant have equivalents in the domain model and the application model. The detection of equivalence is carried out based on the *Repository* content. The steps of the algorithm are explained in details in Section 4.

Duplication-Free Specification: The output of the framework is a specification that does not contain features causing duplication in the SPL.

IV. An Algorithm for Duplication-Free SPL

In this section, we provide the formalization of the basic concepts used in the framework, then we describe the deduplication algorithm.

A. Formalizing the Basic Concepts

Prior to explaining the algorithm, a certain number of predicates must be defined. We denote by D the domain model. PD is the set of variation points of D, and VD is the set of variants of D.

$$PD = \{PD_1, PD_2, \ldots, PD_p\}$$
$$VD = \{VD_1, VD_2, \ldots, VD_q\}$$

Similarly, we denote by A the application model of a derived application. PA is the set of variation points of A, and VA is the set of variants of A.

$$PA = \{PA_1, PA_2, \ldots, PA_s\} \text{ with } s \leq p$$
$$VA = \{VA_1, VA_2, \ldots, VA_t\} \text{ with } t \leq q$$

Thus:

$$PA \subseteq PD \quad \text{and} \quad VA \subseteq VD$$

 ISBN: 978-1-61208-367-4

We denote by S_0 the specification of an evolution, i.e., the set of new features to implement.

$$S_0 = \{F_1, F_2, \ldots, F_n\}$$

P and V are, respectively, the sets of variation points and variants, which correspond to the features defined in S_0.

$$P = \{P_1, P_2, \ldots, P_v\},\ V = \{V_1, V_2, \ldots, V_u\}$$

It has to be noted that P and V are not subsets of PA and VA. In our framework, we consider that a feature in S_0 can be defined as follows: $F_k = (P_i, V_j)$.

B. The Deduplication Algorithm

To implement a new evolution, we propose an evolutionary framework that verifies whether a feature of the specification is duplicated in the model and generates in the end a new verified specification. Figure 2 shows the relationship between the specification of an iteration k-1, the feature F_k and the resulting specification.

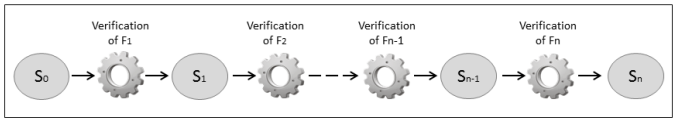


Figure 2. Relationship between S_{k-1}, F_k and S_k.

To verify whether a feature F_k is duplicated or not, the algorithm distinguishes six different cases in each iteration (k-1). These cases are represented in Figure 3.

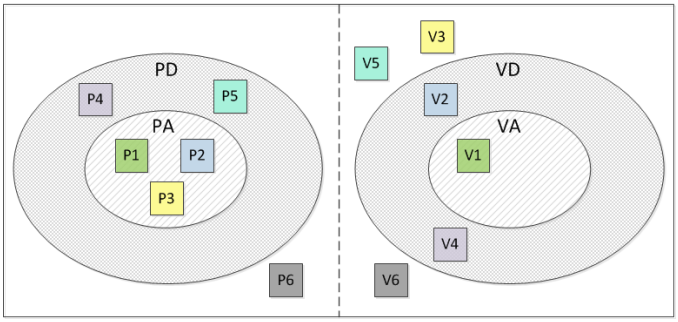


Figure 3. The figure shows the different cases of variants and variation points: Every pair (P_i,V_i) corresponds to a case i.

Case 1: The variation point associated to the feature F_k has an equivalent in PA and the variant has an equivalent in VA. Consequently, the feature is duplicated and must be removed from the specification, but the domain model and the application model do not change.

$$(F_k = (P_i, V_j)) \wedge (Equiv(P_i) \in PA) \wedge (Equiv(V_j) \in VA)$$
$$\Rightarrow \quad S_k = S_{k-1} \setminus \{F_k\}$$

Case 2: The feature F_k consists of adding a new variant to a variation point that has an equivalent in PA, where an equivalent of the variant exists already in VD. Consequently, the feature must be removed from the specification.

$$(F_k = (P_i, V_j)) \wedge (Equiv(P_i) \in PA) \wedge (Equiv(V_j) \in VD \setminus VA)$$
$$\Rightarrow \quad S_k = S_{k-1} \setminus \{F_k\}$$

The variant V_j must be added to the application model.

Case 3: The feature F_k requires adding a new variant to a variation point that has an equivalent in PA, and the variant does not have an equivalent in VD. In this case, we assume that $V = V_S \cup V_G$ where V_S is the set of variants of V that are specific to the business of the application, and V_G is the set of variants of V that are generic. We distinguish two sub-cases:

Case 3.1: If the variant in question belongs to V_S. In this case, the specification does not change and the feature is added directly to the application model.

$$(F_k = (P_i, V_j)) \wedge (Equiv(P_i) \in PA) \wedge\ (Equiv(V_j) = \emptyset) \wedge (V_j \in V_S)$$
$$\Rightarrow \quad S_k = S_{k-1}$$

Case 3.2: If the variant in question belongs to V_G. In this case, the feature is removed from the specification.

$$(F_k = (P_i, V_j)) \wedge (Equiv(P_i) \in PA) \wedge\ (Equiv(V_j) = \emptyset) \wedge (V_j \in V_G)$$
$$\Rightarrow \quad S_k = S_{k-1} \setminus \{F_k\}$$

The variant V_j is added to the domain model and then to the application model.

Case 4: The variation point related to F_k has an equivalent in PD but not in PA, and the variant has an equivalent in VD but not in VA. Consequently, the feature must be removed from the specification.

$$(F_k = (P_i, V_j)) \wedge (Equiv(P_i) \in PD \backslash PA) \wedge (Equiv(V_j) \in VD \backslash VA)$$
$$\Rightarrow \quad S_k = S_{k-1} \setminus \{F_k\}$$

The variation point and the variant must be derived from the domain model and added to the application model.

Case 5: The variation point related to F_k has an equivalent in PD but the variant is new. In this case, the feature is removed from the specification.

$$(F_k = (P_i, V_j)) \wedge (Equiv(P_i) \in PD \setminus PA) \wedge (Equiv(V_j) = \emptyset)$$
$$\Rightarrow \quad S_k = S_{k-1} \setminus \{F_k\}$$

The variant is added to the domain model then to the application model.

Case 6: The variation point related to F_k does not have an equivalent in PD, and the variant does not have an equivalent in VD. In this case, we assume that $P = P_S \cup P_G$ where P_S is the set of variation points of P that are specific to the business of the application, and P_G is the set of variation points of P that are generic. We distinguish two sub-cases:

Case 6.1: If the variation point belongs to P_S, the specification does not change and the feature is added directly to the application model.

$$(F_k = (P_i, V_j)) \wedge (Equiv(P_i) = \emptyset) \wedge (Equiv(P_i) \in P_S)$$
$$\Rightarrow \quad S_k = S_{k-1}$$

Case 6.2: If the variation point belongs to P_G, the feature is removed from the specification, added to the domain model then to the application model.

$$(F_k = (P_i, V_j)) \wedge (Equiv(P_i) = \emptyset) \wedge (Equiv(P_i) \in P_G)$$
$$\Rightarrow \quad S_k = S_{k-1} \setminus \{F_k\}$$

 ISBN: 978-1-61208-367-4

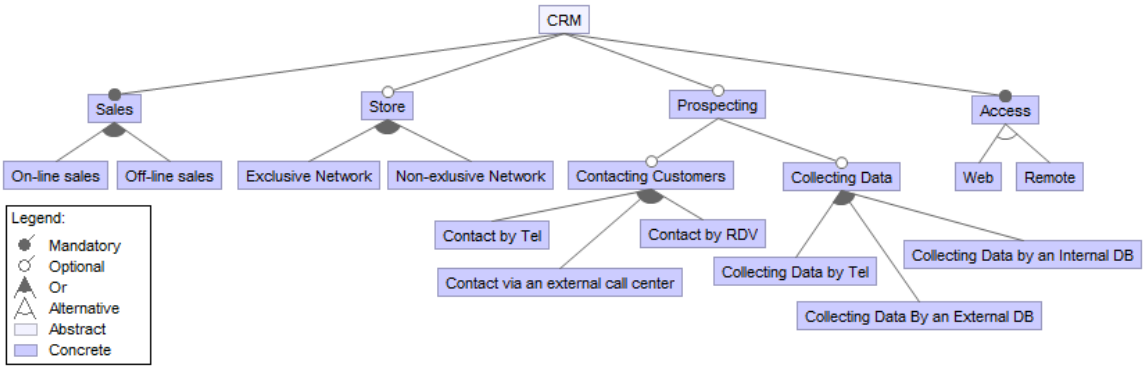


Figure 4. The Domain Feature Model of the CRM.

Result:

In the end, when all the verifications are carried out for all the features of S_0, we obtain S_N. This new specification is a duplication-free specification which contains only the features that have to be implemented directly in the application model.

V. Case Study

To illustrate our framework, we propose a part of a feature model of a CRM (Customer Relationship Management). The Figure 4 depicts the domain feature model of the CRM. The feature model is created using FeatureIDE as mentioned in Section 3.

We consider a derived application with the feature model depicted in Figure 5. Based on the XML source, we consider that the tags "and" correspond to variation points and the tags "feature" correspond to variants.

```
<featureModel chosenLayoutAlgorithm="1">
    <struct>
        <and abstract="true" mandatory="true" name="CRM1">
            <or mandatory="true" name="Sales">
                <feature mandatory="true" name="On-line sales"/>
                <feature mandatory="true" name="Off-line sales"/>
            </or>
            <or name="Store">
                <feature mandatory="true" name="Exclusive network"/>
                <feature mandatory="true" name="Non-exclusive network"/>
            </or>
            <and name="Prospecting">
                <or mandatory="true" name="Collecting Data">
                    <feature mandatory="true" name="Collecting Data by Tel"/>
                    <feature mandatory="true" name="Collecting Data by an external DB"/>
                    <feature mandatory="true" name="Collecting Data by an internal DB"/>
                </or>
            </and>
            <and mandatory="true" name="Access">
                <feature name="Web"/>
            </and>
        </and>
    </struct>
    <constraints/>
    <calculations Auto="true" Constraints="true" Features="true" Redundant="true"/>
    <comments/>
    <featureOrder userDefined="false"/>
</featureModel>
```

Figure 5. The Application Feature Model in XML.

During a new evolution of this application, a number of requirements are demanded by the customer. In our case study, we will take into account only the following requirements:

1) Users can use the application in a disconnected mode.
2) The sector header can contact customers by setting up an appointment.
3) The application must enable users to follow the activity of competitors' shops.
4) The system must manage e-sales.
5) The sector header can generate summary reports in Excel.

We distinguish the following features:

- F_1= (P_1, V_1)= (Connexion, Disconnected mode)
- F_2= (P_2, V_2)= (Contacting Customers, Appointment)
- F_3= (P_3, V_3)= (Shop, Competitor's store)
- F_4= (P_4, V_4)= (Sales, e-sales)
- F_5= (P_5, V_5)= (Reporting, Excel Report)

with $S_0 = \{F_1, F_2, F_3, F_4, F_5\}$.

The list of equivalents of the variation points and variants related to these features is described in Table 1.

TABLE I. The Equivalents of Features

x (VP or V)	Equiv(x)
P_1	Access
V_1	Remote
P_2	Contacting Customers
V_2	RDV
P_3	Store
V_3	$\emptyset$
P_4	Sales
V_4	on-line sales
P_5	$\emptyset$
V_5	$\emptyset$

After applying the verification algorithm to this specification, we came up with the results of Table 2.

*: V_3 is specific to this application, because following the activity of competitors' stores requires a legal authorization, which is not possible for all companies.

**: Generating reports (e.g., in Excel or Word) can be considered as a generic feature, because it can be demanded by other applications.

 ISBN: 978-1-61208-367-4

TABLE II. The Results of the Verification Algorithm

Feature	Verification of VP and V	Case	Result
F_1	$(Equiv(P_1) \in PA) \wedge$ $(Equiv(V_1) \in VD \setminus VA)$	Case 2	$S_1 = S_0 \setminus \{F_1\}$
F_2	$(Equiv(P_2) \in PD \setminus PA) \wedge$ $(Equiv(V_2) \in VD \setminus VA)$	Case 4	$S_2 = S_1 \setminus \{F_2\}$
F_3	$(Equiv(P_3) \in PA) \wedge$ $(Equiv(V_3) = \emptyset) \wedge$ $(V_3 \in V_S)^*$	Case 3.1	$S_3 = S_2$
F_4	$(Equiv(P_4) \in PA)$ $(Equiv(V_4) \in VA)$	Case 1	$S_4 = S_3 \setminus \{F_4\}$
F_5	$(Equiv(P_5) = \emptyset) \wedge$ $(P_5 \in P_G)^{**}$	Case 6.2	$S_5 = S_4 \setminus \{F_5\}$

In the end, we obtain $S_4 = \{F_3\}$, which means that developers have to implement only the feature F_3 in the application model. The other features are all sent to the team maintaining the domain model in order to add the new features and bind the existing ones then re-derive the model.

VI. Related Work

In this section, we provide an overview of the studies most relevant to our work by categorizing them according to the issues addressed in this paper.

Evolution of feature and variability models: In order to reduce complexity and improve the maintenance of variability in large-scale product lines, Dhungana et al. [17] proposes to organize product lines as a set of interrelated model fragments that define the variability of particular parts of the system, and presents a support to semi-automatically merge the different fragments into a complete variability model. The same approach is proposed by Pleuss et al. [18] for feature models. Voelter et al. [19] proposes an approach which consists of separating features in models and composing them by aspect-oriented composition techniques. Cordy et al. [20] defines two particular types of features, regulative features and conservative features, and explains how the addition of these features to the SPL can reduce the overhead of model-checking. The common denominator of the cited studies is that they all consider evolution in domain engineering, while our approach deals with evolution in application engineering.

Model Defects in SPL: Several papers in the literature have addressed model defects caused by SPL Evolution. For example, Guo and Wang [21] proposes to limit the consistency maintenance to the part of the feature model that is affected by the requested change instead of the whole feature model. Romero et al. [22] introduces SPLEmma, a generic evolution framework that enables the validation of controlled SPL evolution by following a Model Driven Engineering approach. This study focused on three main challenges: SPL consistency during evolution, the impact on the family of products and SPL heterogeneity. In [23], Mazo provides a classification of different verification criteria of the product line model that he categorizes into four families: expressiveness criteria, consistency criteria, error-prone criteria and redundancy-free criteria. Redundancy can easily be confused with Duplication, but it is completely different, because Mazo focuses on redundancy of dependencies and not redundancy of features. The same study defines also different conformance checking criteria, among which two features should not have the same name in the same model. This is also different from our approach which is based on equivalence and not only equality of features.

Evolution in application engineering: Carbon et al. [24] presents an empirical study which consists of adapting the planning game to the product line context in order to introduce a lightweight feedback process from application to family engineering at Testo, but it does not provide a general approach that is applicable to all SPLs. Hallsteinsen et al. [25] introduces the concept of Dynamic Software Product Lines (DSPL), which provide mechanisms for binding variation points at runtime in order to keep up with fluctuations in user needs. However, this approach does not explain in details how the variability is managed between application and domain engineering. Thao [26] proposes a versioning system to support the evolution of product line and change propagation between core assets and derived products. But this study also does not provide a method to manage features in application engineering. Our approach is different because it provides a feature-oriented approach to manage the evolution of derived products in a way that insures non-duplication in the SPL feature models.

VII. Conclusion and Future Work

In the literature, many studies have addressed the evolution in SPLs, but the majority of them focused on the domain engineering phase, when application engineering has not been thoroughly discussed. Based on industrial experience, products are also likely to evolve even after their derivation, and this evolution can cause many problems especially duplication in the different artefacts of the product line. In this paper, we provided a framework that deals specifically with duplication in feature models. This framework uses a repository that contains the set of domain features and alternatives of the different concepts of the product line at the aim of verifying the non-duplication of all the features of a new specification. To illustrate the framework, we applied it to a case study from the CRM field. In a future work, we intend to initiate a tool based on the framework architecture, whose objective is to automatize the algorithm of verification and to generate automatically a duplication-free specification that contains only the relevant features to implement.

References

[1] K. Pohl, G. Böckle, and F. Van Der Linden, Software Product Line Engineering Foundations, Principles, and Techniques, Berlin, Germany: Springer-Verlag, 2005.

[2] N. H. Madhavji, J. Fernandez-Ramil, and D. Perry, Software Evolution and Feedback: Theory and Practice, John Wiley & Sons, 2006, ISBN 978-0-470-87180-5.

[3] Y. Xue, Z. Xing, and S. Jarzabek, "Understanding feature evolution in a family of product variants," Proc. WCRE'10, IEEE, Oct. 2010, pp. 109-118.

[4] S. Urli, M. Blay-Fornarino, P. Collet, and S. Mosser, "Using composite feature models to support agile software product line evolution," Proc. 6th International Workshop on Models and Evolution, ACM, Oct. 2012, pp. 21-26.

[5] A. Ahmad, P. Jamshidi, and C. Pahl, "A Framework for Acquisition and Application of Software Architecture Evolution Knowledge," ACM SIGSOFT Software Engineering Notes, vol. 38, no. 5, Sept. 2013, pp. 65-71.

 ISBN: 978-1-61208-367-4

[6] C. Seidl, F. Heidenreich, and U. Assmann, "Co-evolution of models and feature mapping in Software Product Lines," Proc. SPLC'12, ACM, New York, USA, 2012, Vol. 1, pp. 76-85.

[7] P. Clements and L. Northop, Software Product Lines - Practices and Patterns, Boston: Addison-Wesley, 2002.

[8] K. C. Kang et al., "FORM: A feature-oriented reuse method with domain-specific reference architectures," Annals of Software Engineering, vol. 5, no. 1, 1998, pp. 143-168.

[9] C. Salinesi, R. Mazo, O. Djebbi, D. Diaz, A. Lora-Michiels, "Constraints: the Core of Product Line Engineering," In. RCIS'11, IEEE, Guadeloupe-French West Indies, France, May 19-21, 2011, pp. 1-10.

[10] A. Hunt and D. Thomas, The pragmatic programmer: from journeyman to master, Addison-Wesley Professional, 2000.

[11] J. Bosch, Design and use of software architectures: adopting and evolving a product-line approach, New York, USA: ACM Press/Addison-Wesley, 2000.

[12] I. Jacobson, M. Griss, and P. Jonsson, Software Reuse. Architecture, Process and Organization for Business Success, Addison-Wesley, ISBN: 0-201-92476-5, 1997.

[13] S. Creff, "Une modélisation de la variabilité multidimensionnelle pour une évolution incrémentale des lignes de produits," Doctoral dissertation, University of Rennes 1, 2003.

[14] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Technical Report CMU/SEI-90-TR-21, Carnegie Mellon University, Software Engineering Institute, Nov. 1990.

[15] C. Kastner et al., "FeatureIDE: A Tool Framework for Feature-Oriented Software Development," Proc. The 31st International Conference on Software Engineering, 2009, pp. 611-614.

[16] O. Lassila, R. R. Swick, Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 Feb. 1999, http: //www.w3.org/TR/1999/REC-rdf-syntax-19990222/ [retrieved: August, 2014].

[17] D. Dhungana, P. Grünbacher, R. Rabiser, and T. Neumayer, "Structuring the modeling space and supporting evolution in software product line engineering," Journal of Systems and Software, vol. 83, no. 7, 2010, pp. 1108-1122.

[18] A. Pleuss, G. Botterweck, D. Dhungana, A. Polzer, and S. Kowalewski, "Model-driven support for product line evolution on feature level," Journal of Systems and Software, vol. 85, no. 10, 2012, pp. 2261-2274.

[19] M. Voelter and I. Groher, "Product line implementation using aspect-oriented and model-driven software development," Proc. SPLC'07, IEEE, Sept. 2007, pp. 233-242.

[20] M. Cordy, A. Classen, P. Y. Schobbens, P. Heymans, and A. Legay, "Managing evolution in software product lines: A model-checking perspective," Proc. 6th International Workshop on Variability Modeling of Software-Intensive Systems, ACM, Jan. 2012, pp. 183-191.

[21] J. Guo, and Y. Wang, "Towards consistent evolution of feature models," In. Software Product Lines: Going Beyond, Springer Berlin Heidelberg, 2010, pp. 451-455.

[22] D. Romero et al., "SPLEMMA: a generic framework for controlled-evolution of software product lines," Proc. 17th International Software Product Line Conference co-located workshops, ACM, 2013, pp. 59-66.

[23] R. Mazo, "A generic approach for automated verification of product line models," Ph.D. thesis, Pantheon-Sorbonne University, 2011.

[24] R. Carbon, J. Knodel, D. Muthig, and G. Meier, "Providing feedback from application to family engineering-the product line planning game at the testo ag," Proc. SPLC'08, IEEE, Sept. 2008, pp. 180-189.

[25] S. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid, "Dynamic software product lines," Computer, vol. 41, no. 4, 2008, pp. 93-95.

[26] C. Thao, "Managing evolution of software product line," Proc. 34th ICSE'12, IEEE, Jun. 2012, pp. 1619-1621.

 ISBN: 978-1-61208-367-4

Quantum States in Bivalent Logic

Mikael Fridenfalk
Department of Game Design
Uppsala University
Visby, Sweden
mikael.fridenfalk@speldesign.uu.se

***Abstract*—Bivalent or two-valued logic is presently the foundation of logic in mathematics and computer science, and a cornerstone of software development. To address a number of classical logical paradoxes, such as Russell's, multi-valued logic, such as balanced ternary logic has shown to be useful. Current methods lead however to information loss. Thus, to theoretically improve the robustness of bivalent logic, this paper proposes the use of quantum states, followed by an example, where the proposed method is shown to be successful in the solution of a problem that is not directly solvable using contemporary methods.**

Keywords-bivalent logic; propositional logic; quantum state; Russell's paradox; ternary logic

I. INTRODUCTION

On the topics of fundamentals in software development and information modeling, the logical systems today are either typically based on static bivalent logical values, such as true and false, or static fuzzy logic values. This paper does not address quantum logic [1], or quantum computing, where the laws of logic are expanded for application in quantum mechanics, but on the contrary, the application of results from quantum mechanics to classical bivalent logic. In bivalent logic, contradictions, such as Russell's paradox [6], or Epimenides paradox [5] (which may be expressed as "this statement is false"), cause an infinite loop of alternating values such as:

$$\text{true} \rightarrow \text{false} \rightarrow \text{true} \rightarrow \text{false} \ldots \tag{1}$$

Russell's paradox hypothesizes the existence of a set that does not contain itself. Epimenides paradox may be expressed as:

$$x \equiv \neg x \tag{2}$$

In the same way, that resonance may cause instability in control theory, such contradictions may cause instability in machine reasoning, e.g., machine interpretation of propositional logic.

To make a comparison with the field of robotics, it takes in general less effort to program an industrial robot that works in a highly structured environment, than a robot working in an unstructured one, where for instance the precise position, orientation, or the geometry of a workpiece are not always known in advance. Similarly, if we wish to develop machines that are able to handle and solve logical problems in the real world, we need to strive towards the incorporation of a higher level of flexibility in machine information processing, e.g., a higher level of tolerance towards contradictions.

Presently, computer simulations of logical statements (using modern computer languages such as C++ or Java), that include a paradox such as Epimenides or Russell's, tend to either cause the simulation to yield incorrect results, or the program to fall into an infinite loop, why such problems are presently solved manually. The aim of this paper is, therefore, to propose a new method that makes the evaluation of logical statements, that include a paradox (such as Epimenides paradox), intrinsically solvable to computers.

As a brief overview of this paper, Section II succinctly reiterates the state of the art in paradox-tolerant logic. Section III presents a proposal with the aim to further the methods within this field, and in Section IV, the new proposal is verified by computational experiments. Finally, in Section V, an example is provided on the application of the new method in comparison with current ones.

II. RELATED WORK

In context of paradox-tolerant logic, the design of a versatile system was addressed by Lukasiewicz [3] in 1920, using a balanced ternary (three-valued) logic. In this system:

$$\begin{aligned} 1 &\leftarrow \text{true} \\ 0 &\leftarrow \text{unknown} \\ -1 &\leftarrow \text{false} \end{aligned} \tag{3}$$

With the definition of $\neg x$ as $-x$, (2) is solved by:

$$x = -x \Rightarrow x = 0 \tag{4}$$

In addition to negation (not), see Table I, representing Boolean logic [2], other logical connectives may be introduced as well, such as conjunction (and), disjunction (or), implication ($\rightarrow$), and equivalence ($\leftrightarrow$).

In Lukasiewicz logic, conjunction $(x \wedge y)$ may be defined as $\min(x, y)$, and disjunction $(x \vee y)$ as $\max(x, y)$, which, as shown in Table II, produce reasonable results. The downside of this approach is that by using zero to for instance represent a logical wave, we have effectively lost information regarding the phase of this wave for further analysis down the line.

TABLE I. BOOLEAN LOGIC

x	y	$\neg x$	$x \wedge y$	$x \vee y$	$x \rightarrow y$	$x \leftrightarrow y$
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

In this context, another system of interest is the four-valued "Diamond" logic [4], which structurally has many distinct similarities with the proposal presented in this paper. In "Diamond" logic, in addition to the values true and false, two new values are introduced, called i and j, where by definition $i \equiv \neg i$ and $j \equiv \neg j$. This definition resolves the contradiction in (2), but leads from a perspective, to the generation of a new set of contradictions, such as while $i \vee \neg i$ is expected to be a

 ISBN: 978-1-61208-367-4

TABLE II. A BALANCED TERNARY LOGIC WITH $\neg x \equiv -x$, $x \wedge y \equiv \min(x, y)$, $x \vee y \equiv \max(x, y)$, $x \to y \equiv \neg x \wedge y$, AND $x \leftrightarrow y \equiv (x \wedge y) \vee (\neg x \wedge \neg y)$

x	y	$\neg x$	$x \wedge y$	$x \vee y$	$x \to y$	$x \leftrightarrow y$
-1	-1	1	-1	-1	1	1
-1	1	1	-1	1	1	-1
1	-1	-1	-1	1	-1	-1
1	1	-1	1	1	1	1
0	1	0	0	1	1	0
1	0	-1	0	1	0	0
0	-1	0	-1	0	0	0
-1	0	1	-1	0	1	0
0	0	0	0	0	0	0

tautology, since $i \equiv \neg i$, instead $i \vee \neg i \equiv i$, which is caused by information loss.

III. PROPOSAL

In the new proposal, based on two quantum states, ψ and $\bar{\psi}$, using a 2D Boolean vector:

$$\begin{aligned} \text{false} &= 0 = 00_2 \\ \bar{\psi} &= 1 = 01_2 \\ \psi &= 2 = 10_2 \\ \text{true} &= 3 = 11_2 \end{aligned} \tag{5}$$

$$\text{false} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \psi = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad \bar{\psi} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad \text{true} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \tag{6}$$

In this system, all logical connectives are expected to operate element-wise on the 2D Boolean vectors. Thus:

$$\bar{\psi} \equiv \neg\psi, \quad \psi \equiv \neg\bar{\psi} \tag{7}$$

Further on, the equation $x \equiv \neg x$ is here regarded as a discrete wave equation, in essence, similar to the time-dependent Schrodinger equation in quantum mechanics [7]:

$$i\hbar\frac{\partial}{\partial t}\Psi(\mathbf{r}, t) = \left[-\frac{\hbar^2}{2m}\nabla^2 + V(\mathbf{r}, t)\right]\Psi(\mathbf{r}, t) \tag{8}$$

where any wave function that can satisfy this equation is called a "quantum state". A crucial point here is however that while in Lukasiewicz logic, the relation $x \equiv \neg x$ is fully satisfied (and by mere definition in "Diamond" logic), in this new proposal, $x \equiv \neg x$ is only satisfied for ψ and $\bar{\psi}$ by the substitution of x with $\neg x$ on the right hand side of the equation, thereby yielding the solution $x \equiv \neg(\neg x)$. However, in this new approach, the phase of the logical wave is in addition preserved.

IV. RESULTS

According to the results presented in Tables III-IV, the use of quantum states appears to yield a truth table for the selected connectives that preserves the phase of the wave function for further calculations down the line. In this context, it seems however that although the results for $x \leftrightarrow y$ is technically correct (which is equivalent to $x \equiv y$), it would be plausible to define a stronger connective for equivalence as well. In Tables III-IV, "strong" equivalence is denoted as an equivalent sign with four lines instead of three.

TABLE III. FOUR-VALUED LOGIC BASED ON 2D BIVALENT LOGIC, WITH $x \to y \equiv \neg x \wedge y$, AND $x \leftrightarrow y \equiv (x \wedge y) \vee (\neg x \wedge \neg y)$

x	y	$\neg x$	$x \wedge y$	$x \vee y$	$x \to y$	$x \leftrightarrow y$	$\equiv$
00	00	11	00	00	11	11	11
00	01	11	00	01	11	10	00
00	10	11	00	10	11	01	00
00	11	11	00	11	11	00	00
01	00	10	00	01	10	10	00
01	01	10	01	01	11	11	11
01	10	10	00	11	10	00	00
01	11	10	01	11	11	01	00
10	00	01	00	10	01	01	00
10	01	01	00	11	01	00	00
10	10	01	10	10	11	11	11
10	11	01	10	11	11	10	00
11	00	00	00	11	00	00	00
11	01	00	01	11	01	01	00
11	10	00	10	11	10	10	00
11	11	00	11	11	11	11	11

TABLE IV. SAME AS PREVIOUS TABLE, BUT HERE WITH $0 \leftarrow 00_2$, $\psi \leftarrow 01_2$, $\bar{\psi} \leftarrow 10_2$, AND $1 \leftarrow 11_2$

x	y	$\neg x$	$x \wedge y$	$x \vee y$	$x \to y$	$x \leftrightarrow y$	$\equiv$
0	0	1	0	0	1	1	1
0	ψ	1	0	ψ	1	$\bar{\psi}$	0
0	$\bar{\psi}$	1	0	$\bar{\psi}$	1	ψ	0
0	1	1	0	1	1	0	0
ψ	0	$\bar{\psi}$	0	ψ	$\bar{\psi}$	$\bar{\psi}$	0
ψ	ψ	$\bar{\psi}$	ψ	ψ	1	1	1
ψ	$\bar{\psi}$	$\bar{\psi}$	0	1	$\bar{\psi}$	0	0
ψ	1	$\bar{\psi}$	ψ	1	1	ψ	0
$\bar{\psi}$	0	ψ	0	$\bar{\psi}$	ψ	ψ	0
$\bar{\psi}$	ψ	ψ	0	1	ψ	0	0
$\bar{\psi}$	$\bar{\psi}$	ψ	$\bar{\psi}$	$\bar{\psi}$	1	1	1
$\bar{\psi}$	1	ψ	$\bar{\psi}$	1	1	$\bar{\psi}$	0
1	0	0	0	1	0	0	0
1	ψ	0	ψ	1	ψ	ψ	0
1	$\bar{\psi}$	0	$\bar{\psi}$	1	$\bar{\psi}$	$\bar{\psi}$	0
1	1	0	1	1	1	1	1

V. APPLICATION

As an example regarding the new proposal, we consider a problem that is straightforward to figure out for a human, but presently, relatively hard for a machine to solve without any additional assistance.

Problem. A family consists of two parents and two children, A and B. A child that has received the house key will use it to unlock a door for both children. According to the statements made by the parents, $s_1 - s_3$, where the statements are assumed to be mutually synchronized:

s_1: All statements $(s_1 - s_3)$ are false.

s_2: Child A is in possession of the key.

s_3: Child B is not in possession of the key.

The question is hence, are the children able to unlock the door?

Approach 1. Since $s_1 = \neg s_1$ (Epimenides paradox), the use of quantum states, according to the proposal of this paper, yields the solutions: $\{s_2 = \psi, s_3 = \bar{\psi}\}$ and $\{s_2 = \bar{\psi}, s_3 = \psi\}$.

 ISBN: 978-1-61208-367-4

Since $\bar{\psi} \vee \psi \equiv \psi \vee \bar{\psi} \equiv$ true, both solutions yield the correct conclusion that the children are able to unlock the door.

Approach 2. Using Lukasiewicz logic, since $s_1 = \neg s_1$, the (fuzzy-logical) value of s_1 is equivalent to zero, since according to (4), $x = \neg x \Rightarrow x = 0$. Thus, $s_2 = 0$, and since $s_2 = \neg s_3$, thereby, $s_3 = 0$. Further, since $s_2 \vee s_3 = 0$, this yields that we are not able to establish whether any of the children are in possession of the key, and therefore able to unlock the door. No conclusion can thus be drawn.

Approach 3. Using "Diamond" logic, given $s_1 = \neg s_1$, we are able to either define the solution of s_2 as i or j. However, since in the case s_3 is equal to i (or alternatively j), and since $s_2 = \neg s_3$, but $i = \neg i$ (or alternatively $j = \neg j$), this creates a new set of paradoxes, that yield both correct and incorrect solutions. Thus, no univocal conclusion can be drawn.

VI. CONCLUSION

The results of this paper raise questions on the nature of the fundamental building blocks of logic. The logical wave functions ψ and $\bar{\psi}$, as defined here, cannot be directly derived by the static scalar values true or false, but the opposite holds, since true $= \psi \vee \bar{\psi}$, and false $= \psi \wedge \bar{\psi}$. Further on, while static logical values seem to be the root cause of many contradictions in logic, such as Russell's paradox, as shown here, this issue may instead be addressed using quantum states as the building blocks of mathematical logic.

REFERENCES

[1] G. Birkhoff and J. non Neumann, "The Logic of Quantum Mechanics", Annals of Mathematics, Second Series, vol. 37, no. 4, 1936, pp. 823-843.

[2] G. Boole, An Investigation of the Laws of Thought (1854), Watchmaker Publishing, 2010.

[3] L. Borkowski, "On Three-Valued Logic", Selected works by Jan Lukasiewicz, North-Holland, Amsterdam, 1970, pp. 87-88.

[4] N. S. Hellerstein, Diamond: A Paradox Logic, World Scientific, 1997.

[5] D. R. Hofstadter, Gödel, Escher, Bach: An Eternal Golden Braid, Basic Books, 1979.

[6] B. Russell, Principles of Mathematics (1903), W. W. Norton., 1996.

[7] E. Schrodinger, "An Undulatory Theory of the Mechanics of Atoms and Molecules", Physical Review, vol. 28, no. 6, 1926, pp. 1049-1070.

 ISBN: 978-1-61208-367-4

An Approach for Modeling and Transforming Contextually-Aware Software Engineering Workflows

Roy Oberhauser
Computer Science Dept.
Aalen University
Aalen, Germany
roy.oberhauser@htw-aalen.de

Abstract—**Software engineering environments (SEE) face challenges for providing automated guidance for human-centric software development workflows. Among the various issues is the lack of a method to model software engineering (SE) workflows, as commonly used business process modeling notation is generalized and not conducive to context-aware support for SEE, while available SE-specific process model notation, such as the Software & Systems Process Engineering Metamodel, is not executable. The solution approach in this paper simplifies and validates the modeling of SE workflows via graphical modeling capabilities that extend the Software Engineering Workflow Language (SEWL). Model-based transformation of workflow concepts to diverse workflow management systems (WfMS), as well semantic transformation of SE concepts to a contextually-aware process-centered software engineering environment - CoSEEEK, is supported. The results show the viability and practicality of such an approach to graphically model, transform, and enact SE workflows while transforming relevant SE concepts to an ontology that supports contextual guidance capabilities.**

Keywords-process-centered software engineering environments; software engineering environments; software engineering process modeling; software engineering process model transformation.

I. INTRODUCTION

In order to be generally applicable to various software (SW) development projects, most software engineering (SE) process models remain abstract and require tailoring to the specific project, team, and tool environment. Examples of SE process models include VM-XT [1] (specified for all public-sector IT development in Germany) and OpenUP [2]. Typical SE process models are documented in natural languages and are thus not easily executable in an automated form. Executable processes whose sequence can be automated and modeled in a workflow management system (WfMS) are called workflows. SE workflows can cover some sequence of activities and steps related to requirements, design, testing, etc., for instance Activity Flows in VM-XT [3] or workflows in OpenUP [4].

Despite the potential of process-aware information systems (PAIS) to provide automatic process assistance and guidance for humans, this area has lacked satisfactory assistance mechanisms and standards. While process-centered software engineering environments (PCSEEs) have attempted to address this area [5], they remain intrusive, rigid, and inflexible [6], and fail to adequately support the human, creative, and dynamics of SW development. Thus, such systems are ignored or abandoned by SW engineers.

To address this challenge for such human-centric SE processes, we created a PCSEE called the Context-aware Software Engineering Environment Event-driven Framework (CoSEEEK) [7]. Beyond SE tool sensors and other contextual knowledge, it utilizes workflows to understand the process context. That includes knowing which activities a SW engineer performed, which activity is likely currently being worked on, which activity is next, and associates these with SE concepts of project, teams, persons, roles, tools, and artifacts via an ontology and reasoner. While various facets were investigated, including collaboration [8], quality integration [9], and others, we still faced the problem of providing an easy to use way for SW engineers to model and transform SE workflows, integrating SE concepts without vendor-lockin to a specific WfMS or its tools.

Considering possible SE workflow modeling notation, the Software & Systems Process Engineering Metamodel (SPEM) [10] is aimed primarily at defining a domain-specific notation for the documentation of SE processes, and does not completely address issues related to executable SE processes so that automatic support and guidance for software engineers in operational activities can occur. On the other hand, a general workflow language notation such as the Business Process Model and Notation (BPMN) [11], while executable, lacks the inclusion and semantic meaning of various SE domain-specific concepts.

To address the executable SE workflow language gap, our team had created the text-based language SEWL [12] and had targeted the adaptive WfMS AristaFlow [13] and YAWL [14]. This paper contributes various extensions to the original workflow concepts, including: a new graphical representation for SE-specific workflows blending BPMN and SPEM notation; a graphical editor for SE workflows; details on the model-driven generation of tailored artifacts that target the ontology and heterogeneous WfMS support, specifically the common of-the-shelf (COTS) WfMS jBPM [15] and Activiti [16]; and the workflow ontology generator, which addresses the aspect of contextual-awareness support for workflows in conjunction with CoSEEEK.

The summary of the paper is as follows: the next section discusses related work; Section III describes the solution concept; followed by implementation details; Section V presents an evaluation, which is followed by a conclusion.

 ISBN: 978-1-61208-367-4

II. Related Work

SPEM 2.0 [17] was approved without supporting full process enactment. It proposes two possible approaches for enactment: One proposes mapping to project planning tools. However, this does not support automated adaptation to changing project contexts during project execution. The other proposal is to use the Process Behavior package to relate SPEM process elements to external behavior models using proxy classes. Both approaches lack full workflow modeling and executability at the level of BPMN.

Other work related to enactment of SPEM includes eXecutable SPEM (xSPEM) [17]. Process execution is addressed via transformation to the Business Process Execution Language (BPEL), while process validation is addressed via transformation to a Petri net in combination with a model checker. [18] maps SPEM to the UML Extended Workflow Metamodel (UML-EWM) in order to create a concretely executable workflow. [19] and [20] investigate transforming SPEM to BPMN, while [21] maps SPEM to XPDL. xSPIDER ML [22] is an extension profile of SPEM 2.0 to enable process enactment.

The novelty of our solution is that, in contrast to the above approaches, it targets a simple graphical as well as textual SE process language and notation for modeling, blending the strengths of BPMN and SPEM; it concretely generates executable workflows on different WfMS targets; and it generates an OWL-compliant ontology of SE concepts for context-aware PCSEE tooling support. This addresses prior hindrances and challenges for modeling and integrating SE workflows in SEE.

III. Solution

Figure 1 will be used to describe the solution concept. The four SE process phases shown at the top will be referenced in the solution description below. The basis of the solution concept is the SEWL workflow. A SEWL workflow is *modeled*, either with the graphical SEWL editor or a textual editor, and provided as input for the Generator. To *transform* the input, the Generator utilizes various adapters that generate appropriate workflow templates tailored for a specific WfMS while concurrently providing OWL-DL [23] output of semantic concept instances. These templates are then *deployed*. During *operations*, a Process Manager Service abstracts, via an interface, the WfMS-specific management and interaction details for CoSEEEK (thus CoSEEEK doesn't need to be a PAIS but only extend one) and the ontology is referenced internally during *operations* by CoSEEEK.

Ontologies and semantic technology are advantageous in providing a taxonomy for modeled entities and their relations, a vocabulary, and supporting logical statements about entities [24]. Automated consistency checking and interoperability between different applications and agents support reuse.

To support loose-coupling with CoSEEEK, a service-oriented event-driven architecture was used in conjunction with a tuple space [25] composed on top of a native XML database eXist [26]. A Process Manager Service manages and abstracts the peculiarities of each WfMS (such as jBPM and Activiti), interacting via events indirectly with CoSEEEK through the Space.

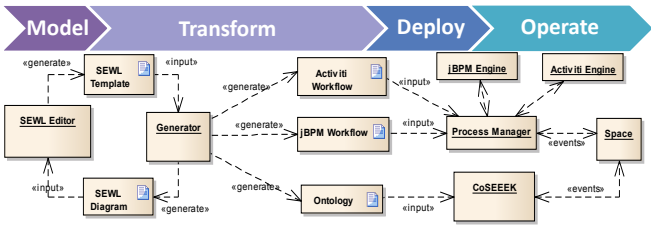


Figure 1. Solution concept.

Model. One primary principle of the approach is that SEWL workflows always remain the reference point of truth. In the process modeling phase, a graphical SEWL Editor assists the process modeler in creating the textual SEWL workflows, which maintain the essence of workflow concepts. Supplemental graphical diagram information (position, font, color, etc.) is retained in separately maintained diagram files, which are kept in sync with the SEWL workflows during the Transform phase. Manual editing of the XML-based SEWL format is thus also possible, however, the Generator will remove all non-applicable elements from the graphical diagrams. Further details are provided in the next section.

Transform. In the Transform phase, workflow templates for the WfMS are generated as shown in Figure 1. The Generator also semantically transforms SE concepts in the workflow to produce an OWL-DL compliant ontology that it utilized for process contextual awareness by CoSEEEK.

Table I shows the mapping of concepts, whereby WUC stands for Work Unit Container and WU for Work Unit. The primary difference between jBPM and Activiti is that in Activiti loops are typically expressed via inclusive gateways, and in jBPM via exclusive gateways. E.g., any concurrent tasks in an SE workflow would be modeled with the BPMN parallelGateway, which activates all branches simultaneously and, when merging, waits for all branches to complete. Most WfMS support such basic features.

TABLE I. Mapping of SE and Workflow Concepts

SEWL	Activiti	jBPM	Ontology
Phase	Service Task + inclusiveGateway	Service Task + exclusiveGateway	WUC + WU
Activity	Service Task	Service Task	WUC + WU
Iteration	Service Task + inclusiveGateway	Service Task + exclusiveGateway	WUC + WU
Task	Service Task	Service Task	WU
Sequence	-	-	-
Parallel	parallelGateway	parallelGateway	-
Loop	inclusiveGateway	exclusiveGateway	-
XOR	exclusiveGateway	exclusiveGateway	-
Roles	-	-	Role Template
Artefacts	-	-	Artefact Template
Variables	-	-	Workflow Variables Template

To address and abstract the integration, communication, and coordination details of the specific WfMS with the

 ISBN: 978-1-61208-367-4

Space, each Activity or Task is represented as a Service Task and during generation wrapped with code that supports the tracking or triggering of the start and finish of an activity or task via event listening and generation.

Deploy. In the process deployment phase, workflows in the WfMS-specific format are deployed into their respective engine and the workflow ontology integrated into CoSEEEK.

Operate. In the process operational phase, the indirect interaction between a Service Task in a WfMS and CoSEEEK via the Space and Process Manager is shown in Figure 2.

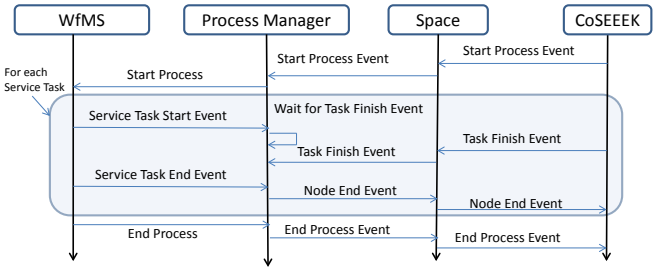


Figure 2. Primary runtime component interaction.

Events (e.g., Task ID 79 start) are written to the Space, and any component can register for events using the Space. As an aside, because all event history is kept in the Space, CoSEEEK components coming online after an absence can determine the context or catch up on any missed events.

IV. IMPLEMENTATION

Details on the implementation will now be discussed. The Graphical Modeling Framework (GMF) and the Eclipse Modeling Framework (EMF), which includes ecore, were utilized. Figure 3 shows a simplified metamodel snippet for implementing the model-driven approach.

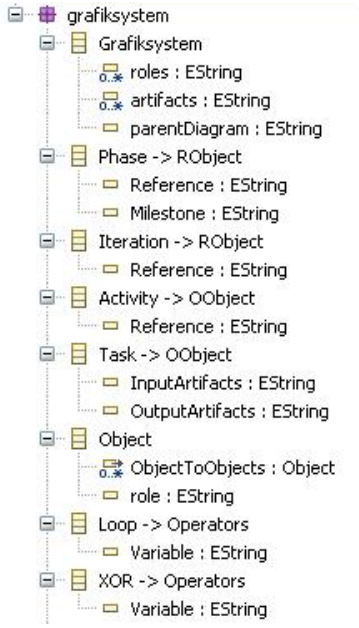


Figure 3. Highly simplified metamodel used in ecore.

Transformation Adapters. Because the transformations are XML-centric, the transformation adapters were coded primarily in Scala. Unique IDs were be generated for every element transformed and its target transformed element. This permits a clear mapping association, useful also for logging. The ontology adapter uses the Jena framework for programmatic ontology access [27] to generate the ontology instances for phases, activities, roles, artefacts, etc.

Figure 4 shows the code generated for the jBPM Service Task, while Figure 5 shows that for Activiti.

```
<task id='2'name='RequestChange'tns:taskName='SEWL Task'>
 <extensionElements>
 <tns:onEntry-script
scriptFormat='http://www.java.com/java'>
  <script>StartEventListener listener = new
StartEventListener();
     kcontext=listener.writeNodeStart(kcontext);</script>
 </tns:onEntry-script>
<tns:onExit-script
scriptFormat='http://www.java.com/java'>
  <script>EndEventListener listener = new
EndEventListener();
           listener.writeNodeEnd(kcontext);</script>
  </tns:onExit-script>
 </extensionElements>
</task>
```

Figure 4. Listing of generated jBPM Service Task.

```
<serviceTask id='RequestChange' name='Request Change'
activity:class='Service'>
 <extensionElements>
  <activity:executionListener event='start'
class='StartEventListener'/>
  <activity:executionListener event='end'
class='EndEventListener'/>
 </extensionElements>
</serviceTask>
```

Figure 5. Listing of generated Activiti Service Task.

The generated OWL output was loaded into Protégé and is shown for a work unit activity in Figure 6. Because the entire XML is very verbose, it is not shown.

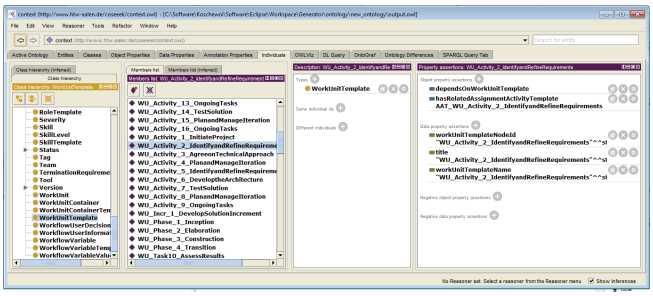


Figure 6: Generated OWL Ontology for CoSEEEK.

SEWL Editor. The textual language supports the specification of SE elements a process may have. Multi-lingual support for referencing the same SE concept instance was implemented, supporting global software development (GSD) processes and their documentation.

The graphical notation is extensible and can be adapted or "skinned" with icons to suit the preferences of the user, which can minimize notation confrontations between different user "tribes", e.g., BPMN purists or SPEM purists.

 ISBN: 978-1-61208-367-4

In order to get the "best of both worlds", the SEWL Editor currently applied a mix of graphical notation as follows:

- SPEM icons for all SE concepts (e.g., phase, activity, iteration, task, role, artefact),
- BPMN icons for process notation, e.g., events, gateways, and connections.

An OpenUP Inception phase workflow in the SEWL Editor is shown in its graphical notation (Figure 7) followed by its textual notation (Figure 8).

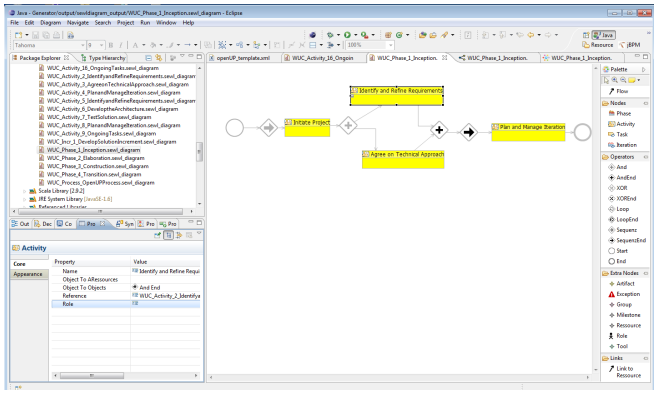


Figure 7. SEWL Editor showing OpenUP Inception Phase diagram.

```
<process base="default_process.xml" xmlns=...>
  <resources>
    <roles>
      <role id="1" name="Analyst" />
      <role id="2" name="Project Manager" />
...
  <elements>
    <element name="phase" base="container">
      <structure>
        <attribute name="repeatable">true</attribute>
      <rules>
        <contains element="activity" />
        <contains element="iteration" />
...
  <artifacts>
    <types/>
    <instances>
      <artifact type="Artifact">Project Plan</artifact>
...
  <tools/>
  <element type="sequence" name="OpenUP Process"
resource="6">
    <element type="phase" name="Inception"
milestone="Lifecycle Objectives">
      <element type="sequence">
        <element type="activity" name="Initiate Project">
          <element type="task" name="Develop Technical
Vision" resource="1">
            <output>
              <parameter name="vision"
tailoring="true">Vision</parameter>
              <parameter name="glossary"
tailoring="true">Glossary</parameter>
...
        <element type="parallel">
          <element type="activity" name="Identify and
Refine Requirements">
            <element type="sequence" resource="1">
...
                <element type="activity" name="Agree on
Technical Approach" resource="4">
...
      <element type="activity" name="Plan and Manage
Iteration" resource="2">
        <element type="sequence">
          <output>...
          </output>
```

Figure 8. Example OpenUP SEWL workflow snippets (end-tags omitted).

To retain graphical data of the layout of nodes and edges, XMI [28] was used. See Figure 12 for an example.

An exemplary subset of the included constraints used to validate the model is listed here, i.e., audit rules. These were implemented in Java to allow usage outside of the GMF.

- Verify phase/activity element has an output and a submodel
- Verify end element has no output
- Verify task does not target iteration/activity/phase
- Verify Loop has LoopEnd, Sequence has SequenceEnd, XOR has XOREnd, And has AndEnd.

V. EVALUATION

The evaluation configuration consisted of an Intel Core 2 Duo CPU 2.26 GHz, 3 GB RAM, Windows XP Pro SP3, JDK 1.6.0-31, Scala 2.91, Activiti 5.8, jBPM 5.2, Eclipse EMT (Helios) SR2. Measurements used `System.nano()`.

Feasibility. As to supporting a broad modeling spectrum, the Eclipse Process Framework (EPF) was used as a reference for modeling Scrum and OpenUP as was an industry partner's internal SE development process. These models were successfully modeled and transformed. Although the complete OpenUP process was modeled, only portions of the Inception Phase are shown below due to space constraints.

Based on the Editor input, the generator was executed and the following output was generated. Since there was no mechanism in the jBPM editor at the time to automatically arrange the elements, all elements are by default placed at the upper left. Thus, Figure 9 was rearranged by hand.

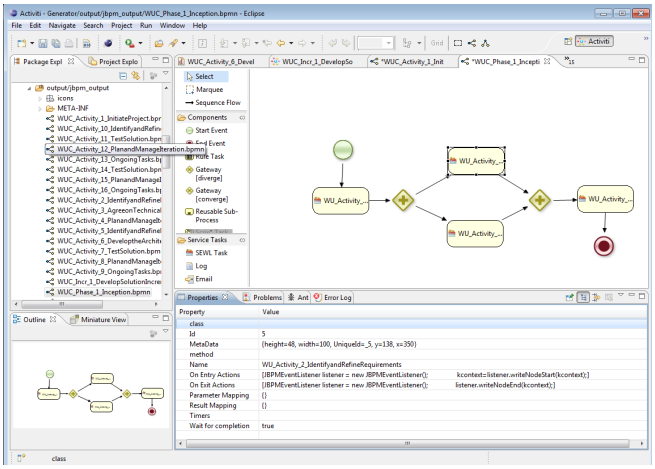


Figure 9. jBPM generated output (arranged later by hand).

A snippet of the corresponding generated output for jBPM is shown in Figure 10 and for Activiti in Figure 11.

Performance. To determine the generator performance, an OpenUP process consisting of a sequence of five nodes was used as the input to the Editor, measuring the performance of each of the generators. For each round, a loop of 1000 generations was averaged. The results are presented in Table II.

 ISBN: 978-1-61208-367-4

TABLE II. GENERATOR PERFORMANCE (IN MILLISEC)

Round	SEWL template	Ontology	Activiti	jBPM	SEWL-Diagram
1	10.3	6701.6	25.2	28.8	65.3
2	10.5	6847.2	24.7	26.6	64.4
3	10.4	6976.9	22.1	27.7	66.4
4	10.1	6901.2	24	25.9	64.2
5	10.4	6945.8	23.2	28.6	65
Avg.	10.3	6917.8	23.8	27.5	65.1

The performance of the generators is satisfactory for typical SE process transformation, except that the verbose OWL generation consumes significant time and should be considered for future optimization.

VI. CONCLUSION AND FUTURE WORK

A solution approach for an easy to use graphical modeling capability for executable SE workflows that can execute on COTS WfMS, while retaining SE semantic information in a separate OWL file for contextually aware PCSEEs, was described conceptually, implemented, and evaluated. The results are indicative that model-based support for transforming SE workflows to common WfMS is both feasible and practical.

Future work includes case studies with industry partners in live settings. Also, bidirectional workflow transformation support between SEWL and an engine-specific workflow format would allow editing in the workflow editor of choice. This entails providing reverse transformation support for engine-specific workflow templates, enabling engine-specific usage of features and editing capabilities via workflow engine-specific editors. For instance, changes made to jBPM and Activiti workflows could be automatically reflected in a SEWL template.

ACKNOWLEDGMENT

The author thanks and acknowledges Vitali Koschewoi for his work on the implementation and diagrams and Gregor Grambow for his assistance with CoSEEEK-related adaptations.

REFERENCES

[1] S. Biffl, D. Winkler, R. Höhn, and H. Wetzel, "Software process improvement in Europe: potential of the new V-modell XT and research issues," Software Process: Improvement and Practice, 11(3), 2006, pp. 229-238.

[2] P. Kroll and B. MacIsaac, Agility and Discipline Made Easy: Practices from OpenUP and RUP. Pearson Education, 2006.

[3] http://v-modell.iabg.de/v-modell-xt-html-english/

[4] http://epf.eclipse.org/wikis/openup/

[5] V. Gruhn, "Process-Centered Software Engineering Environments: A Brief History and Future Challenges," Annals of Software Engineering, 14(1-4), 2002, pp. 363-382.

[6] A. Fuggetta, "Software process: a roadmap," Proc. Conf. on the Future of Software Eng., ACM, May 2000, pp. 25-34.

[7] R. Oberhauser, "Leveraging Semantic Web Computing for Context-Aware Software Engineering Environments," Semantic Web, Gang Wu (ed.), In-Tech, Austria, 2010.

[8] G. Grambow, R. Oberhauser, and M. Reichert, "Enabling Automatic Process-aware Collaboration Support in Software Engineering Projects," Software and Data Technologies (Editors: J. Cordeiro, M. Virvou, B. Shishkov), CCIS 303, Springer Verlag, ISBN 978-3-642-29577-5, 2012, pp. 73-88.

[9] G. Grambow, R. Oberhauser, and M. Reichert, "Contextually Injecting Quality Measures into Software Engineering Processes," the International Journal On Advances in Software, ISSN 1942-2628, vol. 4, no. 1 & 2, 2011, pp. 76-99.

[10] Object Management Group, "Software & Systems Process Engineering Metamodel Specification (SPEM) Version 2.0," Object Management Group, 2008.

[11] Object Management Group, "Business Process Model and Notation (BPMN) Version 2.0," 2011.

[12] G. Grambow, R. Oberhauser, and M. Reichert, "Towards a Workflow Language for Software Engineering," Proc. of the The Tenth IASTED Int'l Conf. on Software Engineering (SE 2011), ISBN 978-0-88986-880-9, ACTA Press, 2011.

[13] P. Dadam et al., "From ADEPT to AristaFlow BPM suite: a research vision has become reality," in Business process management workshops, Springer, Jan. 2010, pp. 529-531.

[14] W. Van Der Aalst and A. Ter Hofstede, "YAWL: yet another workflow language," Information systems, 30(4), 2005, pp. 245-275.

[15] M. Salatino and E. Aliverti, jBPM5 Developer Guide, ISBN 1849516448, Packt Publishing, 2012.

[16] T. Rademakers, "Activiti in Action: Executable business processes in BPMN 2.0," Manning Publications Co., 2012.

[17] R. Bendraou, B. Combemale, X. Crégut, and M. Gervais, "Definition of an Executable SPEM 2.0," In Proc. APSEC 2007, IEEE, 2007, pp. 390-397.

[18] N. Debnath, D. Riesco, M. Cota, J. Garcia Perez-Schofield, and D. Uva, "Supporting the SPEM with a UML Extended Workflow Metamodel," Proc. IEEE Conf. on Computer Systems and Applications, AICCSA, 2006, pp. 1151-1154.

[19] D. Riesco, G. Montejano, N. Debnath, and M. Cota, "Formalizing the Management Automation with Workflow of Software Development Process Based on the SPEM Activities View," Proc. 6th Int'l Conf. on information Technology: New Generations, 2009, pp. 131-136.

[20] M. Perez Cota, D. Riesco, I. Lee, N. Debnath, and G. Montejano, "Transformations from SPEM work sequences to BPMN sequence flows for the automation of software development process," J. Comp. Methods in Sci. and Eng. 10, 1-2S1, (September 2010), 2010, pp. 61-72.

[21] Y. Feng, L. Mingshu, and W. Zhigang, "SPEM2XPDL: Towards SPEM Model Enactment," Proc. of SERP, 2006, pp. 240-245.

[22] C. Portela et al. "xSPIDER ML: Proposal of a Software Processes Enactment Language Compliant with SPEM 2.0," J. of SW Eng. & Applications, 5(6), 2012, pp. 375-384.

[23] D. McGuinness and F. Van Harmelen, "OWL web ontology language overview," W3C recommendation, 2004.

[24] D. Gasevic, D. Djuric, and V. Devedzic, Model driven architecture and ontology development. Springer, 2006.

[25] D. Gelernter, "Generative communication in Linda," ACM Transactions on Programming Languages and Systems, 7(1), 1985, pp. 80-112.

[26] W. Meier, "eXist: An open source native XML database. Web," Web-Services, and Database Systems, LNCS, 2593, 2009, pp. 169-183.

[27] B. McBride, "Jena: a semantic web toolkit," Internet Computing, Nov. 2002, pp. 55-59.

[28] Object Management Group, "MOF 2 XMI Mapping Version 2.4," 2010.

 ISBN: 978-1-61208-367-4

APPENDIX

```
<process processType='Private' isExecutable='true'
id='WUC_Phase_1_Inception' name='Inception'>
 <extensionElements>
  <tns:import name='coseeek.workflow.process
.jbpm.extension.JBPMEventListener'/>
 </extensionElements>
  <startEvent id='_1' name='Start'></startEvent>
...
   <parallelGateway id='_3'
gatewayDirection='Diverging' />
   <parallelGateway id='_4'
gatewayDirection='Converging' />
  <task id='_5'
name='WU_Activity_2_IdentifyandRefineRequirements'
tns:taskName='SEWL Task' >
  <extensionElements>
  <tns:onEntry-script
scriptFormat='http://www.java.com/java'>
  <script>JBPMEventListener listener =
      new JBPMEventListener();
   kcontext=listener.writeNodeStart(kcontext);</script>
  </tns:onEntry-script>
  <tns:onExit-script
     scriptFormat='http://www.java.com/java'>
    <script>JBPMEventListener listener = new
JBPMEventListener();
        listener.writeNodeEnd(kcontext);</script>
    </tns:onExit-script>
  </extensionElements>
  <ioSpecification>
    <inputSet/>
    <outputSet/>
  </ioSpecification>
  </task>
<task id='_6' name='WU_Activity_3_AgreeonTechnicalApp...
```

Figure 10. Example jBPM workflow Snippet.

```
<process id='WUC_Phase_1_Inception' name='Inception'>
 <extensionElements>
    <activiti:executionListener event='start'
class='coseeek.workflow.process.activiti.extension.Proc
essStartEndListener'></activiti:executionListener>
   <activiti:executionListener event='end'
class='coseeek.workflow.process.activiti.extension.Proc
essStartEndListener'></activiti:executionListener>
 </extensionElements>
  <startEvent id='startevent1'
name='Start'></startEvent>
  <endEvent   id='endevent1'  name='End'></endEvent>
    <serviceTask id='WU_Activity_1_InitiateProject'
name='Initiate Project'
activiti:class='coseeek.workflow.process.activiti.exten
sion.DummyService '>
    <extensionElements>
     <activiti:executionListener event='start'
class='coseeek.workflow.process.activiti.extension.Even
tListener'></activiti:executionListener>
     <activiti:executionListener event='end'
class='coseeek.workflow.process.activiti.extension.Even
tListener'></activiti:executionListener>
    </extensionElements>
    </serviceTask>
   <parallelGateway id='parallelGatewayFork1' />
      <serviceTask
id='WU_Activity_2_IdentifyandRefineRequirements'
name='Identify and Refine Requirements'
activiti:class='coseeek.workflow.process.activiti.exten
sion.DummyService'>
      <extensionElements>
       <activiti:executionListener event='start'
class='coseeek.workflow.process.activiti.extension.Even
tListener'></activiti:executionListener>
       <activiti:executionListener event='end'
class='coseeek.workflow.process.activiti.extension.Even
tListener'></activiti:executionListener>
```

Figure 11. Activiti XML Snippet.

```
<graphicsystem:Graphicsystem
xmi:id='WUC_Phase_1_Inception'
parentDiagram='WUC_Process_OpenUPProcess.sewl_diagram'
>
     <newObjects xmi:type='graphicsystem:Start'
xmi:id='startevent1' ObjectToObjects='sequenceStart1'
/>
  <newObjects xmi:type='graphicsystem:Sequenz'
xmi:id='sequenceStart1'
ObjectToObjects='WU_Activity_1_InitiateProject' />
   <newObjects xmi:type='graphicsystem:Activity'
xmi:id='WU_Activity_1_InitiateProject' Name='Initiate
Project'
Reference='WUC_Activity_1_InitiateProject.sewl_diagram'
ObjectToObjects='parallelGatewayStart1' />
...
 </graphicsystem:Graphicsystem>
 <notation:Diagram xmi:id='id_WUC_Phase_1_Inception'
type='SEWL' element='WUC_Phase_1_Inception'
name='Inception.sewl_diagram' measurementUnit='Pixel'>
    <children xmi:type='notation:Shape'
xmi:id='shape_startevent1' type='2043'
element='startevent1'>
...
   </children>
     <children xmi:type='notation:Node'
xmi:id='shape_WU_Activity_1_InitiateProject'
type='2034' element='WU_Activity_1_InitiateProject'>
     <children xmi:type='notation:DecorationNode'
xmi:id='4e841147-2f14-445a-b0b4-30e714be504e'
type='5039'/>
     <children xmi:type='notation:BasicCompartment'
xmi:id='0b62527e-b592-4e3d-a367-541f17843fb9'
type='7011'/>
     <styles xmi:type='notation:DescriptionStyle'
xmi:id='1b9fea72-5856-4be5-9203-1ef5cc58d000'/>
     <styles xmi:type='notation:FontStyle'
xmi:id='3051a516-b9f4-42c6-9698-8072fbe9a301'/>
     <styles xmi:type='notation:LineStyle'
xmi:id='7ea4d238-14fc-4068-a4ce-ed6bb08820af'/>
     <layoutConstraint xmi:type='notation:Bounds'
xmi:id='11135191-6e30-4c7a-a803-dfd437a058bc' x='440'
y='185' />
    </children>
...
 <styles xmi:type='notation:DiagramStyle'
xmi:id='_avAfkaznEeGl_a7M295XCw'/>
  <edges xmi:type='notation:Connector' xmi:id='flow23'
type='4020' source='shape_startevent1'
target='shape_sequenceStart1'>
   <styles xmi:type='notation:FontStyle'
xmi:id='8712763c-8e17-4285-948b-0b78f41f90af' />
     <element xsi:nil='true' />
     <bendpoints xmi:type='notation:RelativeBendpoints'
xmi:id='71805553-c9c1-46ff-8d13-56c6a3ab24fc'
points='[20, 0, -125, 10]$[130, -14, -15, -4]'/>
     <sourceAnchor xmi:type='notation:IdentityAnchor'
xmi:id='63f1b22c-d2fd-408e-9b8a-99044df18ce6' id='EAST'
/>
  <targetAnchor xmi:type='notation:IdentityAnchor'
xmi:id='0fd5db1f-daac-468a-a457-2dcf6bf1ee43'   />
   </edges>
```

Figure 12. Example SEWL diagram XMI code snippet.

 ISBN: 978-1-61208-367-4

A Software Category Model for Graphical User Interface Architectures

Stefan Wendler and Detlef Streitferdt
Software Systems / Process Informatics Department
Ilmenau University of Technology
Ilmenau, Germany
{stefan.wendler, detlef.streitferdt}@tu-ilmenau.de

Abstract — **The development and maintenance of graphical user interfaces (GUI) for business information systems is still affected by software architectures lacking quality. Only basic patterns and few reference architectures are available for GUI development. There exist no standard architectures for reuse. High efforts accumulate for the adaptation of patterns but the resulting architecture quality often does not represent the desired separation of concerns and is hard to maintain. In this work, general GUI architecture design issues are analyzed. The foundation of the analysis is elaborated as a software category tree that represents the common responsibilities within GUI architectures. As result, the major design issues of GUI systems are summarized. To assess other GUI reference architectures, the software category tree may be of value.**

Keywords — GUI software architecture; software architecture; user interface patterns; graphical user interface.

I. INTRODUCTION

A. Motivation

Domain. Business information systems represent a domain that is largely influenced by software architecture considerations. Especially the graphical user interface (GUI) sub-system is likely to induce high efforts [1] for both development and later maintenance. This applies for both standard and individual software systems as a high demand for individually designed GUI systems is actually present.

Problem. However, GUI architectures are not standardized to the required detail, since historically applied patterns have not converged towards a detailed standard architecture governing every responsibility for change. In addition, the higher degree of system integration into business processes demands for explicit implementations of comprehensive requirement artifact types like use cases, tasks and business processes. Those have to be integrated with rather old patterns like MVC [2] and its variants [3], which did not consider such deep and vast requirements basis. Reference architectures [1][4] and several patterns (design and architectural) [5][6] have been suggested, but have not been properly integrated with traceability [7][8] concepts to keep track of requirements. Moreover, GUI frameworks have a large impact on the structure and often cannot be isolated properly to separate technical implementations from domain or project specific requirements.

Consequences. When systems have grown after several maintenance steps, different concerns tend to be mixed up within the GUI architecture the larger the requirements basis is and the more complicated the integrated frameworks are. For instance, application server calls, data handling, task and dialog control flow can no longer clearly separated in the software architecture. Finally, the GUI and application sub-systems cannot be separated easily and the evolution of both depends on each other. Business logic tends to be scattered in the GUI dialogs [9] and the "smart UI antipattern" [10] may become a regular problem. The architecture was layered during design phase, but the encapsulation of components and separation of concerns did not prove in practice [9]. This is maybe due to used frameworks that expect a certain architecture, which alters original design. More likely is the phenomenon that the architecture was based on common patterns and reference architectures that could not be refined in time with respect to desired quality and extensibility. Lastly, the two concluding points from Siedersleben [9] are still of relevance: standardized interfaces between layers are still missing and technical frameworks still dominate the architecture and evolution. Currently, there are even more than three layers in business information systems and the segregation got even more complex.

User interface patterns. Current research is occupied with the integration of a new artifact type in the development of GUI systems. Being based on pattern concepts, user interface patterns (UIPs) have been approached [11][12][13] to facilitate the generative development of GUIs and highly increase the reuse of proven visual and interaction design solutions that originate from descriptive human computer interaction patterns [14][15]. According to the generative nature of these attempts, the development of GUIs shall be shortened by model-based sources that specify both the GUI system's view instances and the coupling between functional related and GUI-system-architecture components.

Current limitations. Currently, there are still design issues within GUI patterns or reference architectures that hinder the evolution and maintenance of existing systems. To establish a target software architecture of high quality for the implementation of UIPs, these issues have to be addressed in the first place. In fact, UIPs need a clear basis of reuse: an architecture with well separated concerns that permits the flexible allocation and exchange of these greater units of design. Whether UIPs will be generated, interpreted or provided by a virtual user interface [16][17] the resulting architecture will be at least as complex as for standard GUIs. So, the common issues in design will prevail and affect UIP based solutions.

B. Objectives

To prepare the integration of UIPs into GUI architecture and at the same time preserve their reusability and variability in different contexts, open issues in GUI architecture development have to be identified and solved. Therefore, our

 ISBN: 978-1-61208-367-4

goal is to provide a detailed analysis of these open design problems. Hence, we will have to identify the re-occurring responsibilities of GUI architectures and their relationships. On that basis, the frequent applied MVC pattern is reviewed. In addition, we will analyze the Quasar client reference architecture [1] that provides more detail than regular patterns and was created especially for the domain.

C. Structure of the Paper

The following section provides descriptions of common patterns and reference architecture considerations for GUIs. In the third section, we will elaborate a general responsibilities model for GUI architectures. In Section IV, the GUI architecture patterns are reviewed. The results are summarized in Section V, before we conclude in Section VI.

II. Related Work

A. Architecture Patterns for Graphical User Interfaces

With the invention of object oriented programming languages, a clear assignment of the cross-cutting concerns, which are common for a GUI dialog, had to be enforced. Eventually, the model view controller pattern was introduced [2] that distinguishes three object types as abstractions to accept defined responsibilities.

In Figure 1, we present a possible architecture application diagram of the MVC pattern. Generally, the MVC pattern promised a separation of concerns, flexibility and even reuse of selected abstractions. From a practical point of view, the classic MVC pattern misses many details that are essential to fulfill these claims. In this regard, the pattern leaves the task to decouple the three abstractions to be solved by the developer. It is noteworthy that the *Controller* is in charge of many responsibilities at once. Both the handling of technical events (*PresentationEvent*) and the initiation of the final processing of data by the application kernel (*ApplicationKernelService)* are governed by the *Controller*. Therefore, this design unit is closely coupled to the *View*, as well as to the *Model*. As far as the *View* is concerned, the structure of the *Model* has to be known to enable the update of defined UI-Controls via *DataRead*.

There exist many sources of the MVC pattern [18][19]. A widely accepted description can be found in [6].

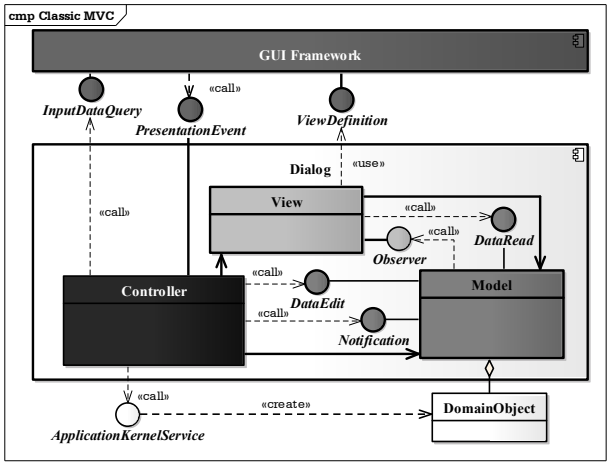


Figure 1. A common MVC architecture pattern variant.

To cope with the close coupling and missing details, several variations of the MVC have been discussed [3][20]. In general, the variations in design differ concerning the distribution of responsibilities among the three abstractions. Several more patterns [5][6][19] occurred that mainly altered the control or introduced new concerns and abstractions. Nevertheless, they fulfill the same purpose of guiding the identification and modularization of classes in object-oriented GUI architectures.

B. Graphical User Interface Event Processing Chain

To be able to discuss the GUI responsibilities with increasing detail, we would like to refer to the conceptual model of event processing within GUI architectures as described by Siedersleben [21]. In Figure 2, a variation of this model is displayed. Thereby, technical events will be emitted from the operation system or later the *GUI Framework* when the user has interacted with a certain GUI element. Within the architecture, the event is either processed or forwarded by the individual components depicted in Figure 2.

It is notable that there is a distinction of events inside the *Dialog* component. For reasons of separation of concerns, and ultimately, better maintenance of systems, the *Presentation* was assigned responsibilities with a closer connection to the technical aspects of the *GUI Framework*. Accordingly, the *Presentation* is in charge of governing the layout of the current *View* and applies changes in layout, e.g., mark the UI-Controls where entered data failed the validation or activate panels when current data state requires for additional inputs. In contrast, the *DialogKernel* is to be kept independent from any technical issues as far as this is possible. So, the latter is assigned to communicate with the *ApplicationKernel* and its components instead.

By flowing all the way from the *Operating System* towards the *Application Component*, a tiny technical event may result in the initiation of greater operations inside the *DialogKernel* or even *ApplicationComponent*. That is why Siedersleben speaks of a "value creation chain" [4][21].

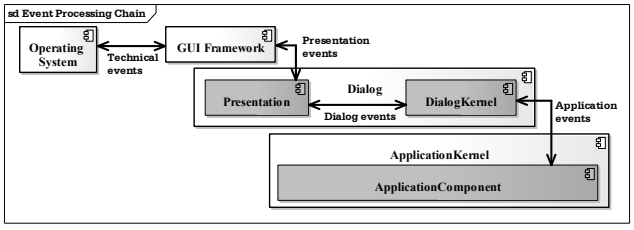


Figure 2. Value creation chain of graphical user interfaces derived from [21].

C. Standard Architecture for Business Information Systems

Siedersleben and Denert tended to the issues of close coupling and a better separation of concerns for GUI architectures in [16]. The main goal of their attempts was to improve the general quality of the software architecture of business information systems. With respect to the GUI, they made suggestions [16] that would prepare the standardization of the architecture of the particular domain.

 ISBN: 978-1-61208-367-4

Quasar. Siedersleben pushed towards further standardization attempts concerning the GUI architecture of business information systems. His efforts culminated in the creation of the quality software architecture (Quasar) [4]. Acclaimed design principles and architectural patterns, as well as the vast usage of interfaces for decoupling in combination with a new instrument for component identification were incorporated into a single software architecture manifest, which was intended to become the domain's standard.

Parts of a reference architecture [1] and the object-relational mapper Quasar Persistence have been published. Conversely, the main ideas of standardization were neglected in [1] and reference architecture elements should fill the gap.

Software categories. As far as the component identification is concerned, so called software categories were introduced. They consist of the five categories *0*, *A*, *T*, *R* and *AT*. *0* designates elements that are reusable in any domain like this is applicable for very basic data types a programming language would offer. *A* software is dedicated to implement a certain domain's requirements, meaning particular functions like the calculation of target costing or the scheduling of production plans for a certain machinery. In contrast, *T* software is responsible for the integration of technical aspects like data bases and GUI frameworks. *R* software is needed whenever a technical data representation has to be converted for processing with *A* software types, e.g., a GUI string type describing a book attribute is converted to a ISSN or ISBN. In fact, *R* software also is *AT* software per definition as both domain specific and technical knowledge or types are mixed up. Thus, *AT* software should be avoided and would be an indicator for the quality of the implementation or architecture. Only the *R* software used for type conversions would be permitted.

GUI reference architecture. Concerning the reference architecture portions of Quasar, the GUI client architecture [1][4] has to be mentioned for the scope of our work. The main parts of that architecture are illustrated by Figure 3 that is derived from [4], since this is the most detailed source available. The interface names in brackets resemble the original but not very descriptive designations. The unique elements of the Quasar client architecture are the following three aspects.

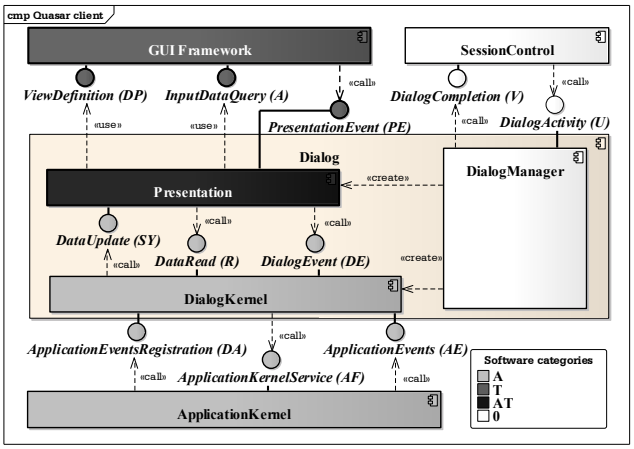


Figure 3. The Quasar client architecture based on [4].

Firstly, there was made a distinction of presentation and application related handling of events; the basic concept of the "value creation chain" introduced in Section II.B was developed further. Thus, there are the two design units *Presentation* and *DialogKernel* that resume original MVC *Controller* tasks besides other ones. The software categories mark both units according to their general responsibilities. The *Presentation* possesses the knowledge how certain data is to be displayed and how the user may trigger events. In contrast, the *DialogKernel* determines what data needs to be displayed and how the application logic should react to the triggered events. The communication between them is exclusively conducted via three *A* type interfaces.

Secondly, the Quasar client introduces relatively detailed interfaces and communication facilities between components compared to other GUI patterns. To be able to fulfill its objectives, the *Presentation* relies on the *ViewDefinition* interface to construct the visual part of the dialog. Via *InputDataQuery*, the current data stored in the technical data model of respective UI-Control instances can be altered or read by the *Presentation*. Events emitted from UI-Control instances are forwarded to the *Presentation* with the operations of *PresentationEvent*.

The interfaces between *Presentation* and *DialogKernel* are mainly concerned with event forwarding and the synchronization of data between both components. In detail, *DialogEvent* is called by the *Presentation* whenever the *DialogKernel* has to be notified of an event relevant for application logic processing, e.g., a command button like OK or a search for available data was initiated. The Quasar client foresees two options for data synchronization. This communication step is essential, since both components possess different knowledge, and thus, work with different data structures, what is marked by the different software categories. Either the *Presentation* could read current data via *DataRead* or the *DialogKernel* would update the *Presentation* by the means of *DataUpdate*. This design shall decouple the application logic from technical aspects found inside *Presentation* and its interfaces for interaction with the current *GUI Framework*.

Thirdly, aspects that are concerned with surrounding components are also described with the Quasar client. These are interfaces dealing with the construction, deletion of dialog instances (*DialogActivity*) and reporting of results (*DialogCompletion*). Furthermore, a DialogKernel can register for notification (*ApplicationEventsRegistration*) about events (*ApplicationEvents*) originated from *ApplicationKernel*. For creation of value relevant for business logic, the interface *ApplicationKernelService* is called by the *DialogKernel*. There are more interfaces available for the coordination of transactions and the checking of permissions via an authorization component. For more details, interface specifications and a dynamic view on the architecture, please consult [1].

III. GENERAL GUI RESPONSIBILITIES MODEL

A. Approach

As the basic GUI patterns and the Quasar client reference architecture are too abstract and general to describe detailed responsibilities required for implementation purposes, we

will establish a fine-grained responsibilities model based on the software category instrument suggested by Quasar. The software categories are intended to refine tasks and fill gaps left open by the available patterns. Thereby, the categories will represent an ideal model with least coupling that allows for planning dependencies among potential units of design.

Consequently, we need to establish a basis for the responsibilities that are regularly discovered in a GUI architecture. Eventually, we follow the approach to investigate on relevant responsibilities mainly from related work, other known sources [2][3][4][6][16][18][19][20][21][22] and own experiences. In fact, we do a decomposition of GUI architectures to rather atomic functions. These functions will be separated and delimited in order to establish a unique software category tree. We examine, what can be solved with *0* or *A* software and what concerns are definitely dependent on GUI framework code.

When common GUI architecture responsibilities have been identified and systematically analyzed concerning their dependencies, the potential interfaces for communication between components or classes can be derived. According to Quasar [4], an interface ideally should be defined on the basis of a software category that serves as a parent for both categories to be linked. Thus, the identification of design units and their interface structure requires some planning.

B. Quasar Software Categories Reviewed

The concept of the Quasar software categories is ambiguous. They promise to be an instrument for component identification and quick software quality assessments. Nevertheless, they were not provided along with a clearly defined method for their proper definition or application.

The software category types defined by Quasar can be applied for the very basic valuation of architectures, since they symbolize a very rudimentary separation of concerns between neutral, domain and technical related concepts. The further and project relevant refinement of the basic categories will eventually lead to a much more powerful representation of design criteria like cohesion and coupling or design principles like modularization as well as hierarchy. In this regard, “concerns” represent heavily abstracted requirements and related functions. Siedersleben [4] states that each software category ideally acts as a representative for a certain delimited topic. Consequently, the preparation of components with the aid of software category trees shall help to create high cohesive and encapsulated design units.

Traceability. On that basis, software categories will be used to judge the purity of traceability-link [7][8] targets, meaning that the artifacts will be examined with respect to their responsibilities. When a target is made up of a mixed category, in the worst case *AT*, then it will be considered either as a model lacking detail or a design that is harder to maintain, since the developers will eventually separate the concerns during implementation by themselves. The latter is a major aspect besides the identification of potential components; that is why we consider software categories as a relevant marker. In sum, software categories can be useful to reduce the complexity while tracing requirements to design: the categories could be kept in order to mark certain design elements inside traceability-metamodels, which are outlined in [8]. Thus, the general or refined responsibilities of design elements will be visible, so traceability-link targets can be more detailed.

A major problem lies in the definition and segregation of software categories. It was not clearly defined what elements drive the creation and delimitation of a software category. According to known sources [4][9], this might either be specialized knowledge how to handle certain algorithms and data structures or dependencies of an entity.

C. Rationale on Software Category Practical Application

Basic software categories. As the software categories are not clearly defined in original sources, we will have to point out how to create new and delimit existing software categories. On the root level, we will comply with Quasar and use the basic categories *0* (white), *A* (light grey), *T* (medium grey with white caption) and *AT* (dark grey with white caption). The basic category *Construction and Configuration* was added to represent the creation of new objects as well as the configuration of interfaces with implementing objects. On the next level, layers and technological boundaries of the application architecture are represented. *Presentation* and *Dialog Logic* were separated as categories according to the event processing of Figure 2. Our aim was to provide a software category tree with separated concerns to describe a complete decomposition of GUI architecture aspects.

As the tree gets more detailed, categories will become very fine grained and embody components, classes or even operations. Since the categories can distinguish components and their dependencies, they could be applicable for the delimitation of the smaller units of design, too.

Category identification. To identify each of the following categories, we applied several rules of thumb. During the analysis of GUI architectures, we derived categories from the different families of operations that regularly occur. In general, these were the definition or modification of new entities or their properties, event triggering or processing, as well as forwarding of both data and events. These kinds of operations occur for different contexts like technical or application related objects of general GUI pattern components that are common for MVC or the Quasar client. The different contexts symbolize certain levels in the software category tree and were derived from reasonable abstractions like application logic, abstract presentation and presentation technology. We distinguished the belonging operations and data structures according to the knowledge and types required for their processing. When operations demanded for the usage of certain types in a context that was not in scope of the originator, then categories were definitely of a mixed kind. In contrast, categories were left pure when interfaces using neutral *0* types could be used for delegations. A hint close to implementation considers what would be the import declarations in a unit of design with respect to Java language. If the import was based on interface types using neutral *0* types, the category would remain pure. The category would be mixed, if the imports will demand for the addition of types defined exclusively in the imported unit of design.

 ISBN: 978-1-61208-367-4

D. Graphical User Interface Software Category Model

The resulting software category tree is depicted in Figure 4 and will be developed in the following paragraphs. It has to be considered that the categories do model dependencies between units of design and no flow of events or algorithms. Although there will be interfaces between categories for later implementation, these cannot be illustrated by the category tree but will be determined concerning the possible type. According to Quasar [4], two different categories may communicate via types that originate from a shared parent category.

The main categories *Application Kernel*, *Dialog Logic*, *Presentation* are *A* category children, since they depend on the individual requirements of a software system.

Presentation. The categories derived from *Presentation* are closely related to the view and controller of the MVC pattern [6] and detail both their responsibilities.

Presentation is marked with FUI (final user interface) [23] given that this category symbolizes the certain knowledge required for creating the specific view part of a given GUI system. This category is further branched into *View Definition* and *Presentation Event Handling*. The involved categories have to comply with project specific dialog specifications and at the same time need to possess knowledge about the types and operations the involved *GUI Framework* offers. Hence, all sub-categories heavily depend on technical aspects. They each constitute a mixed category.

The *View Definition* category is detailed with the responsibilities required for the initial creation of the visual parts of a dialog and the declaration of layout specific elements. We separated the *Layout Definition* and *UI-Control Configuration* as the layout aspects often involve the usage of dedicated objects and operations that considerable differ from the instantiation and configuration of UI-Controls. For the reasons that events require dedicated operations and not all created UI-Controls have to be bound to certain events, the category *Action Binding* was separated as a specialization of the *UI-Control Configuration*.

The *Presentation Event Handling* category serves the task to deal with *Presentation events* according to Figure 2 and is branched into *Presentation Data Handling*, *View State Changes* and *Event Forwarding*. The first child handles both the reading (*Model Data Observer*) and editing (*Model Data Edit*) of dialog data from the *Presentation* perspective. The changes in layout, properties and arrangement of active UI-Control instances during runtime are optional tasks that are embodied by the category *View State Changes* and its children. Certain events cannot be further processed by the visual dialog units, so that they need to notify the next unit in the chain of responsibility. This rationale is based on Figure 2. The required knowledge about the respective events and forwarding commands is encapsulated by *Event Forwarding*.

GUI Framework. As far as the *GUI Framework* is concerned, we decided for the distinction of layout and UI-Control specific knowledge or types. The *UI-Control Library* implements all operations and types that are required for the instantiation of any available UI-Control, the modification of its properties (*UI-Control Properties*) and the definition of its data content (*Technical Data Models*). Often there are various data types with different complexity associated to the available UI-Controls of a framework. They need to be handled by the *Presentation Data Handling* category in order to store and retrieve data in the specific formats like lists, trees, text areas or table grids.

Dialog Logic. The last main category that is to be placed in the vicinity of a dialog is the *Dialog Logic*. Categories that are involved in the data structure definition and its logical processing refine the *Dialog Logic*. The basis of these categories is provided by the Quasar client [1][4] and the model part of the MVC pattern [6]. In analogy to the *Presentation* category, we distinguish the definition of data objects (*Dialog Data Model*) with associated operations and the event handling (*Dialog Event Handling*).

The category *Dialog Data Model* depends on knowledge about the *Domain Data Model* defined by the *Application Kernel* as well as *Data Queries* that may deliver the composition of selected attributes from different entities in order to create new aggregates relevant for display. The *Data Queries* category belongs to the *Application Server Calls* category, which encapsulates knowledge about the available application services, their pre-conditions, invariants and possible results with respect to the dialog logic.

The *Dialog Logic* category graph mostly constitutes pure *A* category refinements. However, the *Data Conversion* category is of mixed character. To define data structures that can be used in close cooperation with the *Application Services*, it needs to know about *Dialog Data Model,* and thus, incorporates its dependencies to the *Data Queries* and *Domain Data Model*. Besides, the *Data Conversion* category has to be aware of the current *Technical Data Models* in order to provide access for *Presentation Data Handling*. The latter has to know about the structure of defined data models (*Dialog Data Model* and *Technical Data Models*) to be able to delegate proper updates in both directions.

Event processing. The entire event processing chain and its association to software categories was challenging; our rationale will be explained as follows. Foremost, logical and presentation states were separated: Application logic tends to be stable (enter data, evaluate, present suggestions, make a choice and confirm), is traced to functional requirements, and thus, should be decoupled from GUI specifications. Although the flow of application logic is unaffected, the GUI and its technology supporting the user in his tasks may be altered several times starting with updated specifications and ending with the deployment of different *GUI Frameworks*. Additionally, the *Presentation* can be further differentiated into abstract visual states that have a close connection to the current application state and technological or concrete presentation states, which implement the former. The latter is translated to GUI UI-Controls via *GUI Framework* and its sub-categories. As result, we identified three major categories for state control to be considered below.

The *Dialog Event Handling* tree governs the application logic part of a dialog and has no concrete visual representations or related tasks. In contrast, it assumes the *Presentation* to maintain appropriate visual representations, but these remain abstract for the *Dialog Event Handling*, e.g., a view for data input is activated, data input was completed or current data leads to another view state for data input.

 ISBN: 978-1-61208-367-4

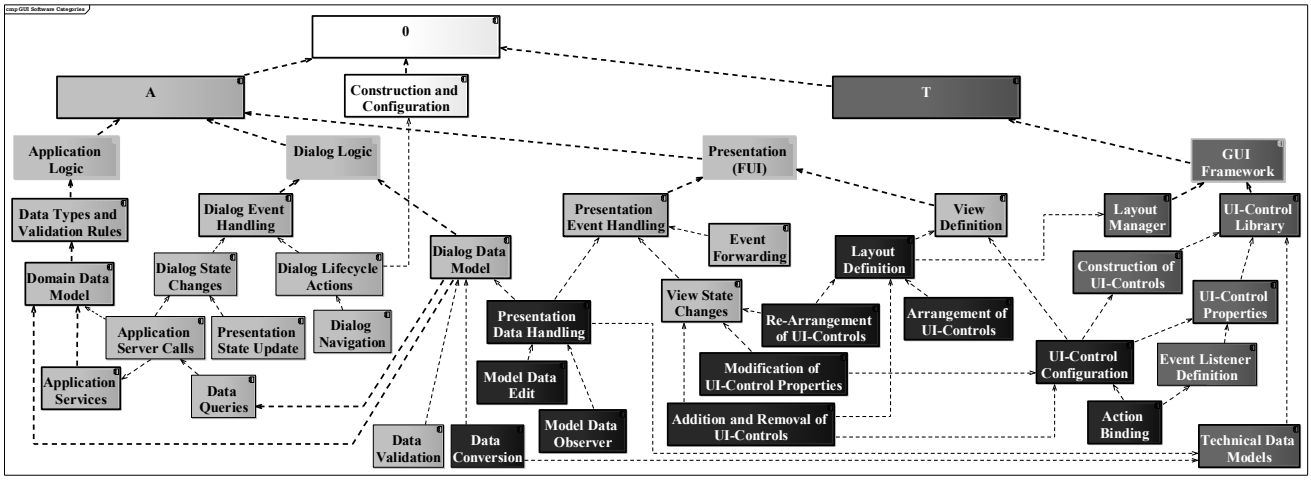


Figure 4. GUI responsibilities arranged as a software category tree.

From the application's perspective, a dialog may adopt different states during runtime. The required knowledge to control these states is represented by the category *Dialog State Changes*. Furthermore, this category is separated into categories, which either interact with the *ApplicationKernel* or the *Presentation*. Both its categories reflect the two general situations that may occur in any dialog: *Application Server Calls* may be initiated or a *Presentation State Update* can be triggered. The parent category *Dialog State Changes* possesses the knowledge how to react in a given situation. Its children are dedicated to solely trigger the required change of state that either addresses the *Application Server* or *Presentation*, which provide the state change execution.

Figure 5 provides an overview of possible interface connections between software categories involved in event processing. Please note that the interfaces need to be of the basic *A* category type as this is the common parent category of the displayed interacting categories.

The general flow of events is the following: initially, the user triggers some events that may be forwarded to *Dialog Event Handling* for further evaluation. Depending on the current state of the dialog, *Dialog Lifecycle Actions* (creation and deletion of dialogs and their objects), *Application Server Calls* (commit a sequence of service calls), a *Dialog Navigation* (change of current view or the instantiation of sub-dialogs) or a *Presentation State Update* (change of the visual representation) may be delegated. In this regard, the key design issue is that the *Presentation* has no knowledge in its sub-categories how to decide on a proper reaction for events relevant for dialog logic. Therefore, the event firstly is forwarded via the topmost interface in Figure 5. Then, the *Dialog Event Handling* evaluates the event and delegates to one of its children, which further delegates to the displayed interfaces in Figure 5 and initiates the final change of state. Concerning the *Presentation State Update* in Figure 5, either a *Dialog Navigation* (separate dialogs or an auxiliary search dialog are instantiated) or *View State Changes* (panels, wizard steps or tabs are switched) are committed via interfaces. In this context, the knowledge when to trigger any of the interface operations is kept in the children of *Dialog Event Handling* with a white border in Figure 5. In contrast, the execution of the respective state change is encapsulated in the categories that implement the interfaces. At last, the state changes are completely decoupled from the point in time when they are requested. Finally, the *Presentation Event Handling* is separated into event processing that is either concerned with data or the visual structure. Mostly the data relevant events can be processed locally by the *Presentation* if no forwarding is registered. However, the *View State Changes* do require the forwarding of events to the *Dialog Event Handling* first, before they can be committed. This is due to the decoupling of view states and their better exchangeability. Moreover, the differentiation of event evaluation, triggering and state change execution supports the reuse and change of views as they are better decoupled from dialog logic components. In this regard, view states are relevant for the *Dialog Logic* but not their concrete appearance, which can be adapted frequently.

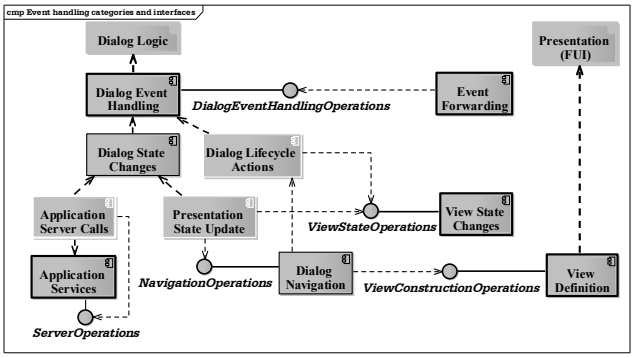


Figure 5. Software categories relevant for event processing and possible interfaces.

IV. Review of GUI Architecture Patterns

In this section, we review the presented GUI patterns of Section II in the light of the elaborated software categories.

A. MVC Variants

For the review of classic GUI architecture patterns, we would like to refer to exemplary and valuable work published in [3] and [20], which is valuable for filling gaps and giving directions for related design decisions. Therein,

options for refinement and customizing MVC based architectures are proposed and discussed. It is still up to the developer to decide on the several choices. In contrast, the Quasar client architecture presents a reference for our domain that already has some refinements incorporated.

Positive aspects. Both patterns and Quasar client share two positive aspects that motivate their application. Firstly, the data storing component does not depend on any other of the components, and so, can independently evolve. Secondly, only one of the components resumes the task to call *ApplicationKernel* services. This aspect eases the design efforts for interfaces and data exchange formats between dialogs and *ApplicationKernel*.

Issues. According to the MVC variants, we see two major issues that will be described as follows.

Separation of concerns. Firstly, the degree of encapsulation and separation of concerns of MVC variants is very limited. There is no variant that is able to reduce the dependencies of all three abstractions altogether. Solely, the distribution of tasks is altered, and so, the visibility among components changes accordingly. In the end, one component will be assigned responsibilities that originate from the two other components as they are defined by classic MVC. Therefore, the component with concentrated tasks tends to be overburdened, and finally, can end up as the bottleneck from a maintenance perspective. Additionally, in certain variants the altering the tasks of the three components may result in a simplification of one component that can only be employed for stereotype tasks. There seems to be no ideal separation of concerns among three components.

In general, there are no hints given how the business logic and its related display can be decoupled. More precisely, the *View* part is directly coupled to the *GUI Framework*. In addition, the knowledge of the *View* has to constitute of how to operate the *GUI Framework* facilities (to construct the visual dialog parts) and what layout as well as what selection, order and arrangement of UI-Controls are needed to embody the domain and the current service in use.

Event differentiation. With regard to the event processing chain of Section II.B, the patterns do not distinguish clearly between events related to technical or application related concerns. In general, a guideline is missing for the decision when to shift between presentation or application related processing of events. So, the developer has to refine the architecture by himself. The reuse may be affected, since the *Controllers* end up processing both types of events for the sake of quick release cycles.

Cohesion. Concerning the identification of possible instances and their proper size, there are hardly any hints when to create new dialog instances or MVC-triads. Thus, the modularization of dialog components is to be done on behalf of the developer. Only the HMVC [24] gives some rudimentary hints. The general size and scope of MVC units is not clear. According to Karagkasidis [20], a *View* may constitute of single UI-Controls (widgets), containers like panels with a certain set of UI-Controls or whole dialogs.

Coupling. With respect to the limited separation of concerns more issues arise. The control of *Presentation* states and the handling of application related events to initiate *ApplicationKernel* service calls are closely coupled to *View* elements. Usually, in many MVC variants *Controller* and *View* maintain a strong dependency where the *Controller* is fully aware of the UI-Controls of the *View*. In fact, both components build an aggregated unit of design that cannot be reused and is harder to maintain. Eventually, a *Controller* can only interact with *Views* that comply with a certain set of states. Whenever the set of UI-Controls changes the possible states of the dialog alter as well, so that the *Controller* implementation may have to be revised each time.

To partly resolve this issue and decouple the *Controller* from application aspects, a developer could revert to the "Model as a Services Façade" [3] MVC variant. The *Model* would be assigned both data structures and related service calls for interaction with the *ApplicationKernel*. This step would raise a comparative discussion as whether it is favorable to build a separate service layer [25] or use the domain model pattern [19] exclusively for the structuring of the *ApplicationKernel*. In our opinion, the *Model* should not act as a service façade, since it would make parts of an *ApplicationKernel* service layer obsolete. According to the resulting dependencies to functional requirements, the traceability-links of use cases or tasks would be scattered among different *Models* and parts of the *ApplicationKernel*. Furthermore, the operations of the Model would be closely coupled to a certain data structure so that a *Model* cannot be easily combined with other application services in the future. Lastly, services should prevail, since there might be other clients besides a particular GUI to rely on services. There are more disadvantages with that solution like the stereotype character of the *Controller* [3], which will only serve a certain pattern of interaction. Thus, the *Model* should only contain data-relevant operations (getter, setter, aggregation and conversion, a state of current selection state, validation) and be reusable with other services. In this regard, the *Model* should act as a mere preparation of a data structure that is useful in the context of *View*.

Summary. The MVC and its derivates require much adaptation in order to be prepared for implementation [22]. The above mentioned issues considerably may have a negative impact the resulting architecture quality. The available patterns are definitely not easy to interpret with respect to the much more responsibilities illustrated by the software category tree in Figure 4.

The tracing of functional requirements to the parts of the GUI which coordinates *ApplicationKernel* will largely depend on the refinements the developers have incorporated. The resulting architectures will be heterogeneous and may add complexity to quickly provide an adapted solution for the particular domain. As long as there are no standard architectures or standardized responsibilities available, the developer is left with many choices that potentially will lead to vast differences in software architecture quality. The improved segregation of software categories in component architectures is goal hard to achieve with available patterns.

B. Quasar Client Reference Architecture

1) General Valuation

The Quasar client architecture provides the most detailed architecture view on GUI systems published so far and can be regarded as a refinement of the common GUI patterns.

Positive aspects. In contrast to the MVC variants, the Quasar client separates *Presentation* and *DialogKernel* as principal dialog components. This separation is the main source for its virtues, since more clearly distinguished *Controller* tasks are achieved. In this regard, the *Presentation* is required to handle technical events and the *DialogKernel* will process application related events in close cooperation with the *ApplicationKernel* services.

States and control. According to Siedersleben [4], the *Presentation* and *DialogKernel* components share a common structure: both possess memory for storing data, states and a control. Thus, both components are able to manage their states independently. A change of layout aspects in the *Presentation* would not affect the *DialogKernel* accordingly.

In theory, the changes of states are implemented in each component individually and can be triggered by *A* typed interfaces that may be designed on the basis of a command [5] pattern [22]. Consequently, the *DialogKernel* does not require knowledge about the inner structure of the *Presentation* and vice versa. Thereby, the *Presentation* may provide a set of operations that alter the layout of a dialog depending on the current content of data collected via *DataUpdate* interface. The triggering of visual state changes on behalf of the *DialogKernel* (*Presentation State Update*) may be possible but is not considered. For instance, a *DialogKernel* was notified via *DialogEvent* that the user has selected an item in a table listing available products. But the product is on back-order, so the *Presentation* should receive the command to display a certain state of the button bar, e.g., deactivate the "add to cart" button. Besides, a *DialogKernel* could be able to coordinate the inputs of a user working with two *Presentations* simultaneously.

2) Traceability-Links to GUI Software Categories

To be able to better valuate the Quasar client architecture, we traced the identified software categories of Section III.D to its structural elements. Figure 6 displays the resulting traceability matrix. The sources for traceability-links constitute software categories of varying detail arranged on the left hand side. Please note that the general parent software categories were excluded, since all child categories are presented in the matrix. On top of the matrix, the traceability-link targets are represented either by the components or interfaces of the Quasar client. Components not relevant as traceability-link targets were excluded.

Interpretation. We need to provide directions about the treatment of interfaces and connected dependencies, which are depicted in Figure 3. A client that imports and calls a foreign interface must have knowledge about the proper usage and sequences of operations. In fact, the deeper and more chained the commands are the more likely is the mixture of categories. Finally, the client will be dependent on the same software category the interface is composed of. This particularly applies to the *Presentation* (obviously an *AT* component) that extensively uses the *GUI Framework* interfaces, which are to be included in the traceability matrix. In contrast, single commands of abstract or stereotype nature like notify calls can be realized with a *0* type interface. Yet, the interfaces pose hard to valuate concepts as they inspire a dynamic view on the architecture like the sequences of commands or flow of algorithms. Ultimately, the interface operations would need further refinement for a final valuation. Partly, the Quasar reference architecture provides basic sequences for interfaces in [1].

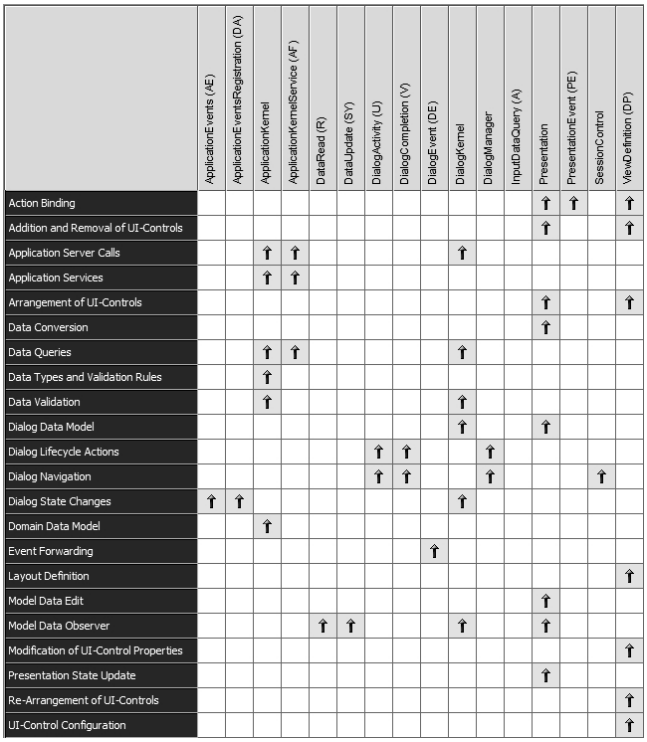


Figure 6. The GUI software categories traced to Quasar client components and interfaces.

Separation of concerns. For the valuation of both cohesion and separation of concerns two directions inside the traceability matrix of Figure 6 have to be considered.

Horizontal. The horizontal direction displays a number of marks for the realization of software categories though components or interfaces. For a high cohesion and well separated concerns, there should be categories realized only by components or interfaces that belong to one unit of design. In sum, *Application Server Calls*, *Data Queries*, *Data Validation*, *Dialog Lifecycle Actions*, *Dialog Navigation* and *Model Data Observer* are realized by several Quasar elements, and thus, different units of design. The first three categories are shared among the *ApplicationKernel* and *DialogKernel*. Thus, the resulting coupling between these design units will largely depend on the refinement of interfaces between both components.

Eventually, a mixture of *A* software categories can be a probable result when no *0* interfaces can be invented. The details of this client and server communication remain an open issue as well as the construction of data queries.

Besides, *Model Data Observer* is presented with two options that are either implemented by the *DialogKernel* (*DataRead*) or *Presentation* (*DataUpdate*). However, the complementary task of *Model Data Edit* is only briefly mentioned. Siedersleben states that the *Presentation* knew about the *DialogKernel* but not vice versa [4]. How the important task of changing dialog data is performed by the *Presentation* and what interfaces are required is left open.

 ISBN: 978-1-61208-367-4

Moreover, *Dialog Lifecycle Actions* are of less importance. They are rather stereotype operations that could be detailed by *0* type software. For the *Dialog Navigation*, there may be missing directions in the Quasar client reference architecture, so that responsibilities have to be refined on behalf of the developer. We wonder how dialog sequences resulting from task model specifications would affect the software category assignments. Maybe the Session cannot be marked as *0* software anymore, since it needs knowledge of the proper sequence of dialogs, which may finally be reused for different task model instances.

Vertical. A further assessment considers the vertical direction that reveals targets with many traceability-links. This can be a marker for lacking detail or even low cohesion. Those targets would take on too many responsibilities at once. There are multiple candidates that awake our attention.

As already stated above, the *ApplicationKernelService* needs further refinement, so that the way how calls and queries are performed by the *DialogKernel* are both detailed and differentiated concerning allowed data types and resulting coupling. Consequently, another major issue is the *DialogKernel* itself. This component is relatively vague in definition, so that tasks like calls to the *ApplicationKernel*, queries, the dialog data definition, data validation and the control of states need to be elaborated from scratch. Concerning functional requirements tracing, the *DialogKernel's* internal structure and state control are important issues that affect the resulting dependencies to requirements. For instance, it has to be decided what portions of a use case will be exclusively realized by the *Application Services* and what parts the *DialogKernel* is in charge of. Above all, the *DialogKernel* is likely to depend to some considerable extent on the *ApplicationKernel* and its *Domain Data Model*. In this regard, it has to be cleared how queries are to be handled from the *Dialog Data Model's* point of view. The *Dialog Data Model* can either be composed of pure entities, which may be embedded as interfaces or data transfer objects, or aggregations that are sourced from selected attributes of several entities retrieved by a query.

Furthermore, the *Presentation* also requires further elaboration in design. Being the complementary part of the *DialogKernel* in a dialog, the *Presentation* is declared as having its own data model in parallel to the *DialogKernel* in order to perform conversions to the *Technical Data Models*. The main data definition is assigned to the *DialogKernel*, since this component is in charge of any data retrieval from the *ApplicationKernel*. How the data related communication (read and edit) besides the notification of updates between *Presentation* and *DialogKernel* is originally intended remains another open issue. In this regard, design decisions on both interfaces and data types as well as their connection to the *Domain Data Model* have to be considered. Moreover, details about the triggering (*Presentation State Update*) and execution of *View State Changes* are missing. This is due to the unclear connection between *Presentation* and *DialogKernel*. When decisions about reactions on events are bound to *Presentation*, logical behavior will be closely coupled to views, so that they are less flexible for change and reuse. In addition, events can only be emitted by view elements and can not be triggered by the evaluation of gathered dialog data alone, since there is no link for the *DialogKernel* to initiate a *View State Change* via *Presentation State Update* when an event was forwarded.

Lastly, the *ViewDefinition* interface and related implementations inside the *Presentation* need more refinement. The coarse grained interface is employed for both handling view states and their initial construction. In this context, a developer would have to decide on how the *DialogKernel* may trigger the visual state changes as a result of its own states defined by *Dialog State Changes*.

3) Summary

Our review of the Quasar client revealed that this reference architecture is more advanced than common GUI patterns. Its main advantage lies in the division of *Controller* tasks among the *Presentation* and *DialogController*, so a better separation of concerns can be achieved. However, this results in increased complexity concerning the number and type of interfaces to be implemented.

In comparison to other architectural patterns, the Quasar client provides more detail and descriptions that give hints to many design decisions, but these are scattered among several sources [4][16][21][22] only available in German language. There was no comprehensive description published, which would provide every needed implementation detail. In the end the Quasar client remains vague with many important issues to solve by individual design decisions. Nevertheless, we learn from the traceability matrix in Figure 6 that there are already hints, which component is to take on what responsibility. In practice, this would yield only a partial improvement with respect to the common GUI patterns. In [1], Haft et al. state that the Quasar client could not be standardized, since most software projects required specific adaptations. The many individual refinements would affect the marking of software categories, so that the purity of them and the separation of concerns may not be maintained as intended. Even the Quasar client assumes that some portions of *AT* software cannot be avoided with conventional architectures relying on invasive frameworks.

To conclude, the Quasar architecture is not suitable for a straight forward implementation. As we see, there are still gaps in the reference architecture and the developer has to incorporate own thoughts in order reach the desired quality architecture. The separation of concerns can be improved with a customized Quasar client architecture, but this largely depends on the skills of the architect. In the end, the Quasar client may be a better, and foremost higher detailed, basis for reuse of architectural knowledge than the MVC variants.

V. RESULTS AND DISCUSSION

1) General Considerations

We derived a software category model that structures the dependencies among common responsibilities of GUI architecture design units. This set of categories can be of aid for the valuation of both the detail and separation of concerns of reference architectures or patterns. In the context of GUI design, the categories resemble different and delimited packages of knowledge, which are used to identify and map components. Later on, the dependencies among the categories will lead the design of interfaces between components [4] to achieve a minimum of coupling. Thus, the

 ISBN: 978-1-61208-367-4

proper distribution of identified categories among design units has an enormous impact on software quality.

Concerning the actual shape of the software categories tree, there might be different structures or aggregations possible (intermediate categories) but the final child elements clearly mark the occurring responsibilities. Currently, concerns like user profiles, additional assistance and authorization are not included. In general, terms in the field of GUI architecture are not used uniformly, so we rely on our category model that provides a clear description of tasks. Furthermore, the software categories may be adapted to fit other domains, since the separation of concerns is essential in most software architectures.

2) Major Issues in GUI Architecture Design

The available architectural patterns differ in structure as well as the encapsulation of concerns. Finally, there is no standardized GUI architecture ready for implementation. This is an issue here but also for mobile devices [26]. We analyzed the differences or missing details of presented architectural patterns and identified three major design issues that may have a considerable impact on GUI maintenance.

Firstly, a design decision has to treat the question what and how much application logic is being processed by a single dialog, or particularly its *DialogKernel*. Thus, the coordination and division of labor between dialog and application related components should clearly define what portions of the event processing chain will just be handled by the *DialogKernel*. As the primary controlling entity of a dialog, the *DialogKernel* acts as a client of the *ApplicationKernel* and its services [4][22]. The architect has to decide how much control flow will be implemented by the client and what operations or services are to be integrated in the controlling object's flow definition. For instance, the business logic can be separated by different layers like services, auxiliary services, domain model entities and data types [27]. The coordination of the various algorithms, which is essential to achieve the goals defined by use cases, can either be performed by the *ApplicationKernel* or the *DialogKernel* may govern the sequence of service calls and their combination. The so called orchestration of services to realize a certain use case is an option for the *DialogKernel*, since this design unit determines the data structure for user interaction. In this context, the *DialogKernel* directly can react to valid user inputs and may decide on the further processing via services or may even trigger corresponding state changes for the *Presentation*. How the latter is to be designed remains an open issue. Siedersleben states that the *ApplicationKernel* components constitute of use case realizations [4]. However, these components would definitely be incomplete use cases realizations, since the latter regularly require much user interaction. To conclude, the question arises how use case realizations are sub-divided among *ApplicationKernel* services (management of data structures and relationships), *DialogKernels* (logic for dialog flow and control of user interaction) and finally *Presentations* (visual part, in- and output UI-Controls). Ultimately, this design decision depends on the navigation structure and whether one *DialogKernel* may control a composition of *Presentation* units or sub-dialogs that form a complete dialog unit for the sake of one use case realization.

This leads us to the second issue that is concerned with the flow of dialog units or navigation among them. Recent research [28][29] investigated on the role of task models for structuring the flow of dialogs. In analogy to the above described issue of division of labor for use case realizations between *ApplicationKernel* and *DialogKernel*, the architect has to decide on the responsibilities of a single *DialogKernel* concerning the flow of dialogs. The question arises what part of the navigation is governed by higher situated components, e.g., a dedicated task controller, and what view changes are in the responsibility of the *DialogKernel*.

Thirdly, the Quasar software categories serve a main purpose to separate application from technical aspects, and thus, avoid *AT* software. As far as the GUI architecture is concerned, we identified two aspects where *AT* software does occur. The *Presentation* communicates with both the *GUI Framework* and *DialogKernel* in order to retrieve data inputs from the user. Eventually, the *Technical Data Models* of the *GUI Framework* and the *Dialog Data Model* have to be converted in the respective formats to enable information exchange. There may be a second conversion necessary between *Dialog Data Model* and *Domain Data Model* when the *DialogKernel* has to use a different data format. Another aspect of *AT* software is the transformation of the *Dialog Data Model* to visual representations, which are constructed by the *Presentation*. Accordingly, the *Presentation* needs to possess knowledge of both the proper selection, arrangement of UI-Controls and the usage, creation of the latter via the specific *GUI Framework* facilities. Besides the first two issues, these two *AT* software aspects can additionally increase maintenance efforts. To solve the third issue, conventional architectures will not suffice and specific designs for additional decoupling have to invented. An initial approach was formulated by Siedersleben and Denert in [16].

3) User Interface Patterns

Before we draw our conclusions, we briefly note how the incorporation of UIPs for the *Presentation* component may resolve the mixture of application and technical aspects. UIPs promise the reuse of visual layout and related interaction. The *Presentation* could be composed of these pattern units and would specify their contents via parameters. The UIP implementations would directly depend on the *GUI Framework* and no longer each *Presentation* unit. Therefore, fewer efforts would have to be spent on programming with *GUI Framework* facilities in the long run when UIPs could be reused extensively. The development could be focused on the *DialogKernel* design issues instead.

To integrate UIPs in the *Presentation*, the differentiated software categories for event processing will be of great value as they prepare the better adaptability and even exchange of *Presentation* units. Responsibilities would be centered in the *DialogKernel* to raise the flexibility of UIPs.

VI. Conclusion and Future Work

The scope of this work is a study of the prevailing issues of GUI architecture design. A software category tree on the basis of Quasar was elaborated, which displays common responsibilities for GUI architectures and their dependencies. With the aid of the software categories, we have analyzed the common GUI MVC pattern and the Quasar client reference

 ISBN: 978-1-61208-367-4

architecture. As result, we identified pattern specific and general issues of relevance for design decisions within GUI architecture development. The herein applied method with a decomposition of software categories and the tracing to an architecture model can be applied for other domains to assess the separation of concerns, cohesion and coupling.

Future work. The findings of this work will influence our further research into the implementation options for UIPs. The Quasar client proved to be the most advanced architecture publicly available. On the basis of the identified issues of that architecture, we will have to develop dedicated solutions to prepare a suitable target architecture for UIPs. We need to further assess the architecture variants outlined in our previous work [17]. The software categories will help us to plan and evaluate possible solutions. Whatever architecture variant will be favored, it definitely needs a software architecture of high quality with well separated concerns to accept UIPs as additional artifacts. The solution must resolve the identified GUI design issues to integrate UIPs in order to reduce the efforts for adaptation of GUIs.

REFERENCES

[1] M. Haft, B. Humm, and J. Siedersleben, "The architect's dilemma – will reference architectures help?," First International Conference on the Quality of Software Architectures (QoSA 2005), Springer LNCS 3712, Sept. 2005, pp. 106-122.

[2] T. Reenskaug, "Thing-Model-View-Editor. An example from a planningsystem," Xerox PARC technical note, 1979.05.12.

[3] S. Alpaev, "Applied MVC patterns. A pattern language," The Computing Research Repository (CoRR), May 2006, http://arxiv.org/abs/cs/0605020, 2014.08.14.

[4] J. Siedersleben, Moderne Softwarearchitektur [Modern software architecture], 1st ed. 2004, corrected reprint. Heidelberg: dpunkt, 2006.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-oriented Software. Reading: Addison-Wesley, 1995.

[6] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stahl, Pattern-Oriented Software Architecture, Volume 1: A System of Patterns. New York: John Wiley & Sons, 1996.

[7] M. Lindvall and K. Sandahl, "Practical implications of traceability," Software - Practice and Experience (SPE), vol. 26, issue 10, Oct. 1996, pp. 1161-1180.

[8] P. Mäder, O. Gotel, and I. Philippow, "Getting back to basics: promoting the use of a traceability information model in practice," The Fifth Workshop on Traceability in Emerging Forms of Software Engineering, IEEE, May 2009, pp. 21-25.

[9] J. Siedersleben, "An interfaced based architecture for business information systems," The Third International Workshop on Software Architecture (ISAW '98), ACM, Nov. 1998, pp. 125-128.

[10] E. Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston, MA: Addison-Wesley, 2004.

[11] A. Wolff, P. Forbrig, A. Dittmar, and D. Reichart, "Tool support for an evolutionary design process using patterns," Workshop on Multi-channel Adaptive Context-sensitive Systems (MAC 06), May 2006, pp. 71-80.

[12] J. Engel and C. Märtin, "PaMGIS: A framework for pattern-based modeling and generation of interactive systems," The Thirteenth International Conference on Human-Computer Interaction (HCII 09), Part I, Springer LNCS 5610, July 2009, pp. 826-835.

[13] K. Breiner, G. Meixner, D. Rombach, M. Seissler, and D. Zühlke, "Efficient generation of ambient intelligent user interfaces," The Fifteenth International Conference on Knowledge-Based and Intelligent Information and Engineering Systems (KES 11), Springer LNCS 6884, Sept. 2011, pp. 136-145.

[14] M. J. Mahemoff and L. J. Johnston, "Pattern languages for usability: an investigation of alternative approaches," The Third Asian Pacific Computer and Human Interaction Conference (APCHI 98), IEEE Computer Society, July 1998, pp. 25-31.

[15] J. Borchers, "A pattern approach to interaction design," Conference on Designing Interactive Systems (DIS 00), ACM, August 2000, pp. 369-378.

[16] J. Siedersleben and E. Denert, "Wie baut man Informationssysteme? Überlegungen zur Standardarchitektur [How to build information systems? Thoughts on a standard architecture]," Informatik Spektrum, vol. 23, issue 4, Aug. 2000, pp. 247-257, doi: 10.1007/s002870000110.

[17] S. Wendler, D. Ammon, T. Kikova, I. Philippow, and D. Streitferdt, "Theoretical and practical implications of user interface patterns applied for the development of graphical user interfaces," International Journal on Advances in Software, vol. 6, nr. 1 & 2, pp. 25-44, 2013, IARIA, ISSN: 1942-2628, http://www.iariajournals.org/software/.

[18] J. Dunkel and A. Holitschke, Softwarearchitektur für die Praxis [Software architecture for practice]. Berlin: Springer, 2003.

[19] M. Fowler, Patterns of Enterprise Application Architecture. New Jersey: Addison-Wesley Professional, 2003.

[20] A. Karagkasidis, "Developing GUI applications: architectural patterns revisited," The Thirteenth Annual European Conference on Pattern Languages of Programming (EuroPLoP 2008), CEUR-WS.org, July 2008.

[21] J. Siedersleben (ed.), "Quasar: Die sd&m Standardarchitektur [Quasar: The standard architecture of sd&m]. Part 2, 2. edn. sd&m Research: 2003.

[22] M. Haft and B. Olleck, "Komponentenbasierte Client-Architektur [Component-based client architecture]," Informatik Spektrum, vol. 30, issue 3, June 2007, pp. 143-158, doi: 10.1007/s00287-007-0153-9.

[23] J. Vanderdonckt, "A MDA-compliant environment for developing user interfaces of information systems," The Seventeenth International Conference on Advanced Information Systems Engineering (CAiSE 2005), Springer LNCS 3520, June 2005, pp. 16-31.

[24] J. Cai, R. Kapila, and G. Pal, "HMVC: The layered pattern for developing strong client tiers," JavaWorld Magazine, http://www.javaworld.com/javaworld/jw-07-2000/jw-0721-hmvc.html (2000), 2014.08.14.

[25] R. Stafford, "Service Layer," in [19].

[26] K. Sokolova, M. Lemercier, and L. Garcia, "Android passive MVC: a novel architecture model for the android application development," The Fifth International Conference on Pervasive Patterns and Applications (PATTERNS 2013), IARIA, May 27 - June 1 2013, pp 7-12.

[27] S. Wendler and D. Streitferdt, "An analysis of the generative user interface pattern structure," International Journal On Advances in Intelligent Systems, vol. 7, nr. 1 & 2, pp. 113-134, 2014, IARIA, ISSN: 1942-2679, http://www.iariajournals.org/intelligent_systems/index.html.

[28] F. Radeke and P. Forbrig, "Patterns in task-based modeling of user interfaces," The Sixth International Workshop on Task Models and Diagrams for Users Interface Design (TAMODIA 07), Springer LNCS 4849, Nov. 2007, pp. 184-197.

[29] V. Tran, M. Kolp, J. Vanderdonckt, and Y. Wautelet, "Using task and data models for user interface declarative generation," The Twelfth International Conference on Enterprise Information Systems (ICEIS 2010), vol. 5, HCI, SciTePress, June 2010, pp. 155-160.

 ISBN: 978-1-61208-367-4

The Impact of User Interface Patterns on Software Architecture Quality

Stefan Wendler and Detlef Streitferdt
Software Systems / Process Informatics Department
Ilmenau University of Technology
Ilmenau, Germany
{stefan.wendler, detlef.streitferdt}@tu-ilmenau.de

Abstract — **Current research suggests user interface patterns (UIPs) to lessen efforts for the development and adaptation of graphical user interfaces (GUI). UIPs shall enable the reuse of both layout and interaction definitions that can be instantiated for any desired context. Most approaches are based on generative development. However, no details about target architectures or examples that prove the variability and proper structuring of UIP artifacts have been published yet. According to conventional GUI architecture development, major design decisions have to be solved individually, since no standard architectures are presently available. This applies to UIP based solutions as well, so that the target architectures are both hard to establish and maintain. On the basis of a general GUI responsibilities model, prevailing GUI design issues will be analyzed according to their impact on UIP based solutions. Furthermore, UIP specific responsibilities are identified and modeled as a software category graph. With this work, the implementation options of UIP architectures are discussed. Finally, we draft a possible solution architecture on the basis of these generalized concerns.**

Keywords — user interface patterns; model-based user interface development; HCI patterns; user interface generation; GUI software architecture; graphical user interface.

I. INTRODUCTION

A. Motivation

Domain. Nowadays, business processes build on the vast support of business information systems. These systems have to realize a large set of requirements that presume a multitude of services that are requested to handle thousands of data sets with a clearly defined stereotype structure. Depending on the domain and specialization of business processes, standard software for customizing to specific requirements or software that has to be developed individually remain as options for their IT-support.

Individual GUIs. Regardless of the chosen solution, the demand for individually designed graphical user interfaces (GUI) has to be considered as a great impact on software architecture. Proven human computer interaction (HCI) patterns [1] enable usability traits that can be essential for both user acceptance and productivity. Therefore, those patterns are to be applied to the context of dialogs, which will be coupled to the application services and data structures the users need to interact with according to business process definitions. In this context, standard software quickly is pushed to its limits concerning customization options for individual dialogs. As far as individual software is concerned, generative and model-based development has greatly advanced with respect to the creation of stereotype structures within a software architecture.

User interface patterns. However, the development and maintenance of GUI dialogs still implies high efforts. To achieve a higher efficiency on the basis of increased reuse, HCI patterns are to be formalized in order to apply them for effective generation of dialog views. On that basis, user interface patterns [2][3] (UIPs) have emerged that shall model essential HCI pattern structures. In addition, the new kind of pattern offers parameterization options in order to apply the corresponding HCI pattern to any suitable context. In sum, the application of UIPs promises many feats for future generative development

Limitations. Currently, two major issues obstruct the vast deployment of UIPs.

Primarily, the UIP concept itself has not gained sufficient maturity: the current state of formalization for UIPs is still not adequate with respect to UIP variability requirements [3], which are essential for a general application of UIPs as versatile patterns. The design of a dedicated UIP language could be initiated as an option and already was attempted [4] or is work in progress [5]. Nevertheless, high efforts are to be considered for that approach.

Besides, UIPs require a software architecture of high quality due to their high reusability and variability traits. The architecture has to be composed of a stable set of components with standardized interface structures to allow the reuse of UIPs within and among different software projects. Thus, UIPs need to be integrated into an encapsulated structure within the GUI sub-system, so that the realization of workflows, functional requirements and related application components is not affected. Ultimately, UIPs have to be decoupled from their application context. The current research into GUI architectures does not provide such an architecture and approaches that are already based on UIPs have not published details of target architectures yet. We will briefly reason about that architecture concerns.

Architecture concerns. Available patterns [6] and related sources [7][8] provide valuable aspects for design decisions, but they are rather isolated and have to be integrated into one comprehensive reference architecture that allows the seamless integration of UIPs. In this respect, common MVC variants and the Quasar client reference architecture [9] are too general in concept [10], so that major design decisions are still to be elaborated in order to allow the effective deployment of UIPs.

Moreover, the technical GUI frameworks already define some architecture constraints for action- and data-binding, as well as control facilities. So, the architect has to find ways to limit their influence on the variability of UIPs, otherwise

 ISBN: 978-1-61208-367-4

UIPs would only be applicable in a certain technical environment.

Ultimately, the development of a high quality software architecture on the basis of a clearly defined requirements structure takes considerable time and has to mature by the experience gained during several projects. Often budgets are just as high to barely exceed break-even or reuse is not envisioned or planned [11], and so, hardly any efforts remain to build and refine reference architectures in the aftermath. In the end, this tasks remains for academic research.

B. Objectives

With our previous work on UIPs [12][3][13] and general GUI architecture responsibilities [10], we have a solid foundation to approach the above introduced UIP- and GUI-architecture limitations.

Firstly, we have to consider that UIP based solutions heavily rely on a pre-defined architecture to accommodate code structures build from both the pattern and instance or configuration information. Consequently, we have to analyze the major GUI design decisions and identify additional responsibilities required for the implementation of UIPs.

Since model-based approaches are already work in progress, we will have to critically discuss the principal UIP implementation architectures. Accordingly, we will criticize the general formalization approach and argue for an alternative solution. As a consequence, we draft a suitable GUI reference architecture based on the new UIP concerns.

C. Structure of the Paper

In the following section, related work that is relevant for our objectives is presented. The third section presents our analysis of the impacts UIP based solutions have on the general GUI design issues. In addition, a software category model is described that details the UIP specific responsibilities of a GUI architecture. In Section IV, the principal UIP implementation alternatives are discussed. A UIP based architecture is drafted in Section V, before we present our results in Section VI. Finally, we draw our conclusions and state future work in Section VII.

II. Related Work

A. Standard GUI Architecture for Business Information Systems

Siedersleben and Denert [14] already tended to the missing GUI architecture standardization issue outlined in the introduction. To enable a more effective design with respect to separation of concerns and increased adaptability to changes, business information systems had to be designed on the basis of a standard architecture, which would incorporate a defined set of patterns and interfaces.

One of those patterns of the envisioned standard architecture was the Virtual User Interface (VUI) that is depicted in Figure 1. The VUI should allow a developer to implement dialogs with a high independence from the rendering *GUI Framework*. In detail, a *Dialog* and its events should be implemented with the aid of the technical independent, abstract interfaces *WidgetBuilder* and *EventListener* rather than using certain interfaces and objects of the imported *GUI Framework* directly. The primary goal was to preserve the interchangeability of the *GUI Framework* without affecting existing dialog implementations. Solely the component *Virtual User Interface* would interact directly with the *GUI Framework*, and thus, would depend on technological aspects.

The basic concepts worked as follows. A *Dialog* would create and even adapt its view at runtime with the operations provided by *WidgetBuilder*. The *VUI* could be delegated by the *Dialog* in order to construct and configure a new status and button bar inside a specified frame. Moreover, the *VUI* would notify the *Dialog* via the interface *EventListener* when events would have been induced by UI-Controls. More details are not known.

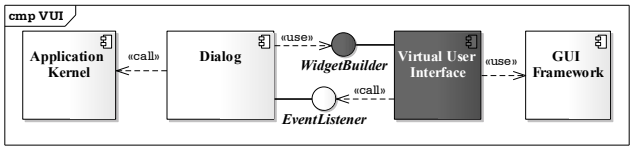


Figure 1. Virtual user interface architecture as introduced in [14].

B. GUI Software Categories and Design Issues

No further ideas for the standardization of an architecture for the domain have been published. A GUI reference architecture [15][9] (Quasar client) and a concept for the identification of components as well as their interface design [15] were presented instead. The latter was based on software categories that would mark the responsibilities and dependencies of a given component. These categories could be used to valuate the cohesion of a given modular structure according to the separation of concerns principle of design.

In [10], we applied the software category concept for the identification and delimitation of general GUI responsibilities. In this regard, the common MVC variants [16][17] and the Quasar client architecture [9][15] were considered both for analysis, and besides other sources, the derivation of software categories. The resulting software category hierarchy and their dependencies are illustrated in Figure 2. The related sources mostly separate the very basic categories *0* (a programming languages' reusable foundations), *A* (application, domain) and *T* (technical aspects, frameworks) without any refinement. Being based on these general software categories, each refined software category of Figure 2 represents the knowledge required for implementing the operations, their proper sequence and required data structures for the respective responsibility they are entitled with.

As a result of our analysis, we derived three major GUI design issues. Firstly, the architect has to decide on how much application control flow is assigned to GUI dialogs and how they coordinate the interaction with the application kernel. This would also influence the application related event processing, and in particular, the update of presentation view states due to changed application data (another view for a certain use case step). Secondly, for the navigation among dialogs and flow of the sub-dialogs a dedicated controlling component has to be allocated. Thirdly, the transformation of application aspects like data models and the visual representation of domain model entities have to be solved. In this regard, a tight coupling to technical frameworks should be limited. For details, [10] can be consulted.

 ISBN: 978-1-61208-367-4

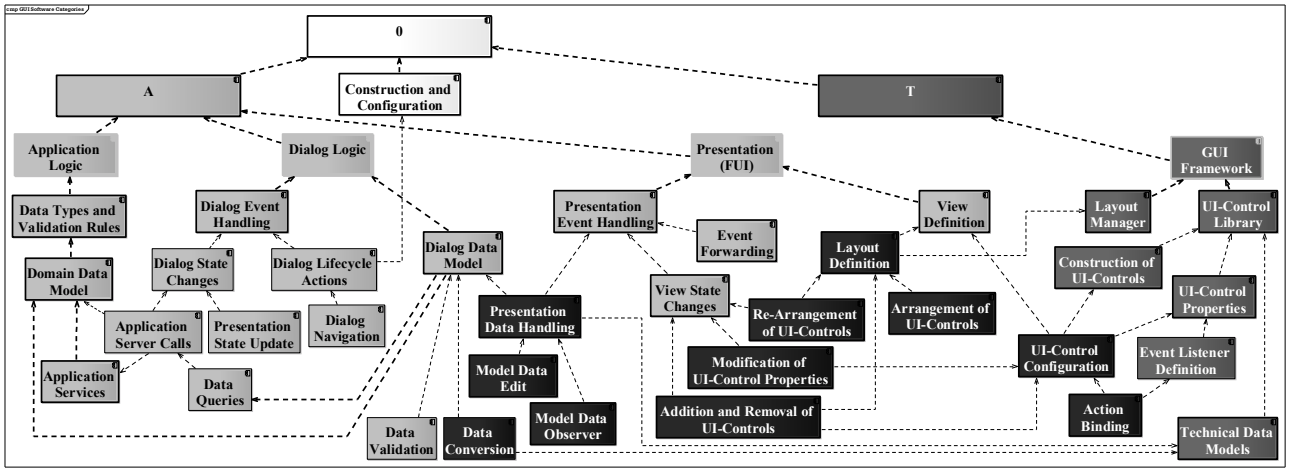


Figure 2. General GUI responsibilities modeled via software categories.

C. User Interface Pattern Aspects

Past work in the field of HCI resulted in the combination of the specification of reusable GUI visual design and interaction solutions with a pattern-based description. Several pattern languages emerged [18]. Current research is trying to exploit these patterns for the automated generation of GUIs. As a consequence, UIPs are based on the idea to formalize HCI visual designs into software patterns that can be reused in any desired application context.

In [12], we elaborated the theoretical and practical implications of that kind of pattern applied within the general transformations from domain requirements to a final user interface specification. As result, UIPs are very promising for bridging the gap between pure requirements and potential GUI specifications, since they define many aspects like used UI-Controls and their interaction designs. Particularly, the latter can be reused to imagine and prototype GUIs of high usability. Moreover, we presented and discussed general architectures for the practical application of UIPs.

With our contributions [2][19], first criteria and aspects for the UIPs to be deployed for variable GUI dialog generation were introduced. Based thereupon, we formed a drafted definition of that particular artifact. The UIP requirements were considerably refined in [3] by the description of an influence factor model. Particularly, the abilities of UIPs were defined by the three aspects *view* (visual elements, layout), *interaction* (view states, events and data-binding) and *control* (composition and interaction of UIPs, binding to application relevant events).

Lastly, the UIPs aspects were further detailed by an analysis model [13], which was derived from the impacts of the influence factor model and describes the resulting structure of a UIP. The elaborated structure could be positively evaluated with UIP examples illustrated by object models. With that last step, basic foundations of UIPs and many aspects that are essential for the formal expression of a UIP are available now.

D. Model-based Frameworks on the Basis of UIPs

Past research has put considerable efforts into the deployment of UIPs or closely related patterns within model-based developments processes.

The University of Rostock [20] mainly worked on the derivation of dialogs from task models and included UIP-like artifacts called PICs (pattern instance components) for the generation of final views. A dedicated UIP formalization language on the basis of UsiXML [21] called UsiPXML [4] was created in parallel. A continuation is not known.

The University of Augsburg presented research into UIPs with the introduction of an own modeling framework called PaMGIS [5]. To express UIPs, a dedicated DTD was partly presented in [5]. The work is still in progress.

The University of Kaiserslautern focused on the application of UIPs for the domain of production environments [22] and sought a way of enabling GUI devices to be able to adapt their view at runtime [23]. In their approach, UIML [24] as a basic GUI specification language is used and augmented with a pattern interface and configuration facilities to be interpreted at runtime. There are only few details of the modeling framework [25] published.

In sum, all approaches suggest individual modeling frameworks that rely on specific formalization formats of UIPs and produce different outputs. A detailed review of these approaches compared to our UIP requirements model is provided in [3].

III. User Interface Pattern Architecture Impacts and Responsibilities

A. Approach and GUI Software Categories

Due to the prevailing issues in GUI architecture design, the development of a new reference architecture for UIPs is most likely to be approached. The interfaces between components need to be harmonized to fit UIPs as reusable entities that may be exchanged to allow the quick adaptation of GUI dialogs. In addition, the event processing has to be prepared to allow the exchange and re-configuration of UIP instances. Finally, UIPs will require a new quality of the software architecture with additional responsibilities.

Category refinement. For the design of such a reference architecture, it is of the essence to consider the separation of concerns. To prepare a proper component identification in this context, the software categories presented in Section II.B will be of great value. They already incorporate the basic

 ISBN: 978-1-61208-367-4

separation of application and technology as requested in [9][11][14][15] as far as possible. In addition, they feature fine-grained refinements of both areas of knowledge. This is essential to avoid coarse grained software categories that concentrate too many responsibilities and would not improve traceability. With coarse grained software categories the component identification would not guarantee separated concerns, since components eventually would have to be refined on the fly during implementation. So, the creation of traceability-links would rely on the coarse grained architecture models, and most likely, would not result in a detailed impact analysis. In contrast, the categories of Figure 2 were separated to a more fine grained level that is able to guide the decisions for GUI design issues. Especially the event processing was differentiated concerning the context (*Presentation*, *DialogLogic*), triggers (*Dialog Event Handling* and children) and execution (*Application Server Calls*, *Dialog Navigation*, *View State Changes*) of state changes. Furthermore, our analysis of the Quasar client with the aid of the software categories in [10] revealed several open issues that were due to lacking details or cohesion.

B. The Impact of User Interface Patterns on GUI Design Issues

We discuss how UIPs will impact the GUI design decisions and ultimately affect the identified responsibilities.

A-T-separation. Foremost, UIPs will stress the separation of *A* and *T* categories due to their variability: If UIPs are bound to a certain *GUI Framework*, they will be virtually rendered useless for architectures employing other technical environments.

Besides this very basic separation, an additional separation has to be considered between *Dialog Logic* and *Presentation* design. To allow the quick adaptation of dialogs, the logical part of a dialog (dialog kernel) has to be able to interact with a presentation that may be altered in design frequently. The former should not be affected when the presentation design was changed to an alternative set of UIP instances. For instance, two large panels for editing data in a single dialog were re-structured into a dialog featuring two tabs instead. Thus, both logic and visual dialog parts have to be decoupled for the adaptation of UIPs.

Flow of application logic. Concerning the division of labor between application and dialog kernel, UIPs need a single basis for coupling of their generic events to context specific behavior. The *OutputActions* of a UIP [13] should be processed centralized by a single component like the dialog kernel to preserve the exchangeability of UIPs emitting those events from the variable presentation part. In this regard, the category *Presentation State Update* gains importance and shall enable a dialog kernel to govern visual changes regardless of the concrete *Presentation* implementation and its UIP instances. The further rationale is to decouple application-independent UIP events from application specific interpretation and processing. In principle, a UIP may be configured to emit an event that may be interpreted very differently in various dialog kernel contexts. With respect to UIP combinations that form one *Presentation* in interaction with the dialog kernel, the individual UIPs have to be kept independent from each other to allow for flexible combinations. One UIP shall not limit the flexibility and change of states of another. In return, a UIP needs a standardized interface to application related artifacts for *Event Forwarding*.

Besides event handling, this also applies for the *Data-binding* impact [3] UIPs require. Obviously, the dialog kernel will become a direct interaction partner for both events and data of a number of UIP instances that are to be integrated together instead of a single view or *Presentation* unit. Therefore, the context for UIPs has to be kept rather isolated from application kernel components, what allows versatile combinations between both. Finally, it has to be considered to centralize the flow of interaction specified by use case models [26] in order to keep an implicit but recognizable connection between UIPs and those functional requirements. In this regard, the dialog kernel may serve once again as central unit that coordinates both *Application Server Calls* and *Presentation State Updates*. The latter establish the implicit connection between UIPs, their states or instantiation and use case steps.

Navigation. The scope of UIPs can be limited to visual elements within dialogs or can even span entire dialog types and their navigation. The different UIP abstractions are symbolized by the various pattern types defined in model-based frameworks [5][23][27]. For the implementation of UIPs that trigger and design dialog navigation like wizards or tabs [28], a dedicated component will be needed that translates the events emitted from these UIPs into the desired change of views or dialogs. The rationale for the centralization is that UIP instance combinations can be very versatile, though UIPs only define the UI-Controls that can be assigned to trigger navigation events. Finally, the evaluation of these events has to be governed by the same component that implements the navigation for non-UIP dialogs in order to allow the seamless integration of UIPs with ordinary dialogs. According to the software category tree, the respective responsibility belongs to the task set of *Dialog Event Handling*, since the navigation is restricted by validation results. For instance, each wizard dialog needs valid inputs to allow the navigation to the next step.

UI-Control set. A further aspect raised by UIPs is the availability of certain UI-Control implementations. For every domain or project, a range of certain UIPs is of relevance. These are to be defined on the concrete user interface (CUI) level of abstraction [12] with reference to [29]. Therefore, the UIPs have to be transformed into UI-Control compositions on the final user interface (FUI) level [29] of abstraction. The CUI based implementation of UIPs ensures their platform independent application and decouples them from *GUI Framework* specific concepts. However, UIP basic elements must be covered by the favored GUI frameworks. For instance, one cannot expect to develop UIPs on the basis of Java AWT due to the very limited set of UI-Controls.

 ISBN: 978-1-61208-367-4

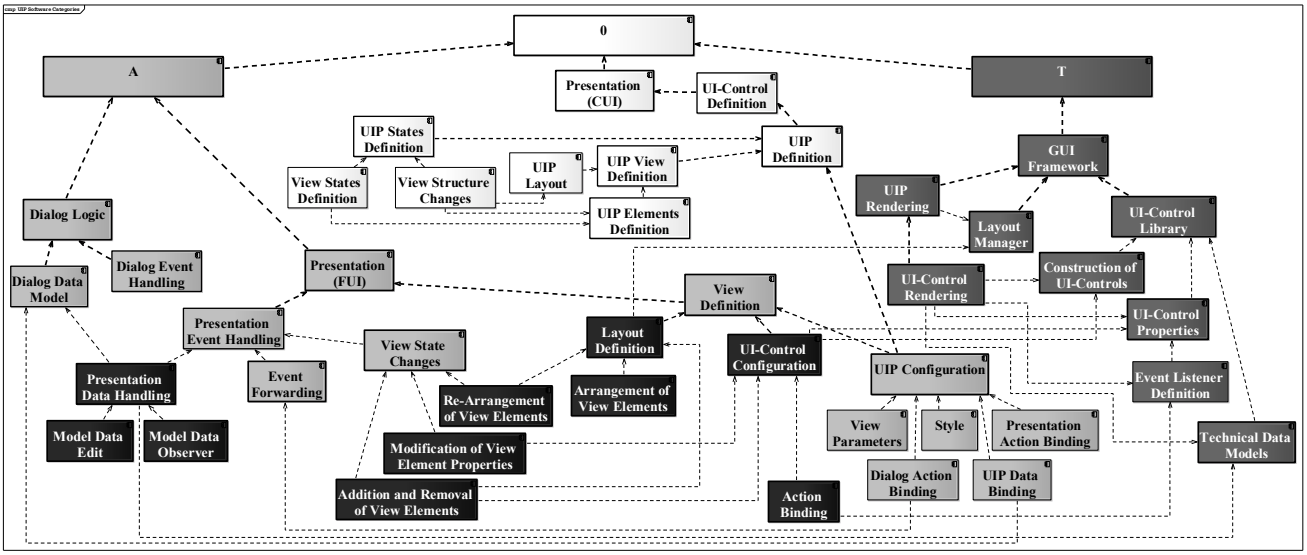


Figure 3. GUI software categories enhanced with UIP based responsibilities.

C. User Interface Pattern Responsibilities Model

From the basic foundations of our previous work and the above mentioned design aspects, the UIP related responsibilities of a GUI architecture will be developed.

Factor model. The influence factor model for UIPs [3] describes additional requirements besides the general GUI architecture responsibilities. As a consequence, the software categories have to be enhanced to reflect the configuration and variability aspects of UIPs. The resulting software category tree is essential for the identification of components of the UIP implementation, the planning of their dependencies and the consideration of UIP requirements. Finally, the category model will translate the factor impacts to comprehensive categories of software component design.

Analysis model. The UIP analysis model [13] represents detailed structures that refine the impacts of the influence factor model. In detail, the analysis model describes coupling points between GUI architecture and UIP configuration facilities, basic structures for UIP units and detailed parameters for visual and behavior aspects. According to the software category identification, the information is useful to mark dependencies to existing basic GUI responsibilities. Afterwards, the analysis model will drive the design of the final UIP representations rather than the software categories.

A-T-separation. The enhanced software category tree is depicted in Figure 3. It is apparent that the UIP software category tree is largely influenced by the mandatory A-T-separation impact. This results into a new hierarchy of *0* software categories. The *Presentation (CUI)* defines the view elements to be reusable in any project. In detail, the *UI-Control Definition* is essential to provide a generally available set of UI-Controls as building blocks for the definition of UIP units. Therefore, *UIP Definition* is dependent on the former. The other categories that refine *UIP Definition* are directly derived from the impacts of the influence factor model. In general, the *0* based categories only define the reusable elements, their properties and abstract behavior, but no final user interface is implemented.

Furthermore, the new *0* category elements can be declared to be used for the *Presentation (FUI)* via *UIP Configuration*, but the rendering has to be implemented for the chosen platform individually. Therefore, the *T* software categories *UI-Control* and *UIP Rendering* were added. These depend on *GUI Framework* sub-categories like this is the case for the conventional *Presentation (FUI)* categories [10].

The ordinary *Presentation (FUI)* composition usually consists of four basic operations: The construction of new UI-Controls and the setting of their properties (*UI-Control Configuration*), the addition of the new UI-Control to a superior container like a panel or frame (*Arrangement of View Elements*) and the optional definition of an event listener (*Action Binding*). All these operations are bundled into respective *AT* software categories, which directly combine domain specific knowledge (content, properties and placement) with technical operations (construction, auxiliary objects like layout constraints or scroll panes) later in code.

When UIPs are instantiated, the above basic operations are distributed among reusable pattern information (*UIP Definition, UI-Control Definition*), context specific configuration (*UIP Configuration*) and the technical rendering (*UIP* and *UI-Control Rendering*). The *Renderings* do not depend on the respective *Definitions*, since they are solely in charge of either the construction of new UI-Controls (*UI-Control Rendering*) or the arrangement of a specific layout (*UIP Rendering*). For that purpose, the *Definitions* define and use basic parameterized operations for their content that are finally implemented by the respective *Renderings*. The *Definitions* contain operations of higher order and the *Renderings* consist of rather atomic ones, hence a *Definition* command will be translated by the *Renderings* into multiple *GUI Framework* calls. Thus, the technical details that are usually present in the ordinary *View Definition* sub-categories are encapsulated by the *Renderings*. The *UI-Control Rendering* will be called with complete information based on the UIP instance parameters, so that only complete units can be created with the *Definition* commands. In this context, the *UIP Configuration* gathers

 ISBN: 978-1-61208-367-4

parameter data with knowledge about the *UIP Definition* (what parameters are exactly present). After the parameters have been configured, they are passed to the *UIP Definition*, which contains all necessary commands in proper order for UIP instantiation and finally delegates the *Renderings*. The latter will implement the abstract operations of the *UIP* or *UI-Control Definition*. The *UIP* and *UI-Control Rendering* are in analogy to *View Definition* sub-categories *UI-Control Configuration* and *Layout Definition*: UI-Controls do not define the gross layout. This instead is a task of the higher situated category *Layout Definition*. Accordingly, there is the distinction between both *Renderings*.

This order of operations is not obvious from the software category graph, since this kind of modeling lacks a runtime or sequence view. In this regard, the dependencies of Figure 3 do not describe calling sequences. The dynamic aspects of calls and the purity of categories will be better visible with component diagrams that describe interfaces and feature the assignment of categories to components. Eventually, the component and interface modeling will refine and verify the software category model.

Changes were applied to sub-categories of *View Definition* and *View State Changes* of *Presentation (FUI)* to reflect the widened set of available view elements. The responsibilities now apply both for UI-Controls and UIP instances.

Flow of application logic. Concerning the flow of application logic and the integration of UIPs, the *Dialog Action Binding* is dependent on *Event Forwarding*, since the ordinary facilities of the event processing chain [10] have to be reused by UIPs and shall not be influenced by a conflicting solution. With respect to the Quasar Client reference architecture [15], the dialog kernel is likely to receive UIP application events in parallel to events from ordinary *Presentations*. In principle, for any UI-Controls of a UIP *PresentationEvents* can be defined [13]. During configuration of UIP instances, application relevant *OutputActions* can be assigned to these events [13]. To preserve this variable binding of UIPs and their events to application behavior (*Dialog Logic*), UIPs have to be decoupled from application logic. This is achieved by the following concepts. Initially, *PresentationEvents* have to be configured for UI-Control instances to be deployed within a particular UIP instance. These can be used to model a trigger for either *ViewStateActions* or *ViewStructureActions* that may add or remove view elements during runtime [13]. In addition, particular *PresentationEvents* can be linked to *OutputActions* that are relevant for application logic (*Dialog Event Handling*) outside the UIP instance. A further decoupling is achieved by the separation of *Event Forwarding* (notification of an event), the decision of a proper reaction by *Dialog Event Handling*, and finally, the implementation of resulting state changes, e.g., *View State Changes* of the *Presentation* [10]. In other words, two states of knowledge are separated: Firstly, what and when events are to be reported. Here, the *OutputActions* mark those events of relevance. Secondly, how will be the reaction implemented that corresponds to reported events. Ultimately, this separation of concerns will allow either the integration of UIPs or ordinary *Presentations* as sender of events relevant for application behavior. This design will allow the versatile configuration of UIPs and their exchangeability. However, a dedicated receiver is essential, which processes events and interacts with application components.

Navigation. In analogy, the navigation design has to follow the same concept: a UIP may emit events that are translated into resulting navigation by a dedicated component. Both concepts preserve the later exchangeability of UIP instances, and thus, allow the decoupling of *Presentation (FUI)* and *Dialog Logic*.

Summary. Finally, UIPs require a GUI architecture that provides a working infrastructure for *Application Server Calls*, *Dialog Navigation*, platform-specific implementations of their UI-Controls and facilities for event as well as data binding. In fact, UIPs can only be applied to describe certain configurations. Thus, the situational meaning of this information is out of the scope of reusable *UIP Definitions* but is to be processed by existing GUI components based on common responsibilities like those modeled in Figure 2. Accordingly, UIP solutions will be based on many common GUI software categories. Therefore, the basic GUI design decisions presented in [10] and discussed here for UIPs in the previous section have to be solved prior to any UIP implementation. Ultimately, UIPs need an elaborate GUI reference architecture with a clearly defined component structure as suggested by the software category model of Figure 2: the new responsibilities are merely enhancements with many dependencies to the basic categories. Particularly, the differentiated categories for event processing [10] will be an essential basis for flexible UIP integration.

The categories partly may be too fine grained, but these serve their purpose better than coarse grained ones that lead to less cohesion and less effective tracing. In contrast, the fine grained categories may later serve as units for lower level design like classes or even operations.

Anyway, the control aspects of UIPs [3] are not modeled here besides *Dialog Action Binding*. This is due to these aspects are cross-cutting concerns that need further elaboration on the basis of detailed examples.

D. Virtual User Interface reviewed

To solve the A-T-separation and maintain the purity of software categories, the virtual user interface from Section II.A is considered.

The main idea of Siedersleben and Denert [14] was to abstract common operations needed for the communication with technical GUI components into lean and easy to reuse interfaces that would considerable simplify the usage of complex APIs or associated frameworks. This concept could yield several benefits when applied for UIP instantiation.

Firstly, the VUI allows the implementation of styleguide rules [14] and other related layout specifics. Therefore, the created layout corresponds to specified rules and could be augmented by standard presentation elements like status or button bars whenever UI-Controls or entire dialogs are requested to be build. This scope of pre-defined GUI layout and selection of UI-Controls can be extended to enable the creation of UIPs. For given UIPs, common UI-Control elements or even nested UIPs that occur regularly as children can be realized as ready to reuse compositions as well.

 ISBN: 978-1-61208-367-4

Secondly, the VUI is worth a consideration for UIPs, since its suggested way of dialog implementation conforms to the concrete user interface model level (CUI) of the Cameleon reference model [29]. This level of GUI modeling foresees certain types of UI-Controls, which may be a common intersection of the ones that are offered by several popular GUI frameworks. Besides, these UI-Controls remain independent from a platform specific implementation as this is the main emphasis of the VUI. Ideally, available UIP implementations could be reused together with alternative GUI-Frameworks.

Thirdly, when the main idea behind the VUI and its interface operations are fully complied with, both basic UI-Control creation and UIP instantiation will have to be realized resulting in a hierarchy of GUI building operations. Therefore, the basic VUI interfaces are relevant for the bottom-up composition of UIPs. Additionally, non-UIP based dialogs could be created at the same time.

However, no details and implementations have been published for the VUI yet. It remains as a general pattern only and solutions must be drafted individually. In particular, the involved interfaces have to be standardized for a GUI system and its dialog types. This step is of the essence, since it permits the reuse of reoccurring functionality such as the creation of views with common UI-Controls and their binding to events. To conclude, the essential elements the GUI system presentation component will constitute of have to be abstracted very clearly and completely in order to provide a CUI level model suitable for the domain.

IV. Discussion of User Interface Pattern Implementation Options

A. Criteria

The principal architecture concepts for UIP implementation were briefly outlined in [12]. Accordingly, we distinct the two concepts of model-based generation and a solution being based on the virtual user interface architecture described in Sections II.A and III.D. We will discuss these alternatives in the light of the GUI design issues and more recent state of the art. The criteria to be considered are presented below.

The primary criterion is the UIP formalization and its completeness. All structural properties and variability aspects of these patterns [13] should be expressed by the chosen notation. Finally, UIPs should be expressed by a CUI model to preserve the platform independent specification.

The second criterion considers the target architecture and respective assumptions. In detail, the integration of UIP instances with other architecture artifacts, which affects the major GUI design issues, is reflected. Since UIPs mostly assume presentation responsibilities, their interface to application logic has to be lean to ensure a variable presentation without affecting application components. For the sake of adaptability, the *Dialog Logic* and associated navigation control should be decoupled from specific UIP instances, too. To preserve the option to integrate non UIP-based dialogs, the decoupling is essential.

A third criterion considers the required tools, and lastly, the coupling to a certain platform and potential reuse of concepts are considered.

B. Model-based Generation

Formalization of UIPs. The model-based frameworks introduced in Section II.D employ their specific format for expressing UIPs for the generation of GUIs. It is noteworthy that the capabilities of the applied notations are not published completely or mentioned at all. In addition, no detailed examples that proof the variability, composition ability and reuse of formalized UIPs have been published yet. Therefore, the maturity of the generation based UIP approaches surveyed in [3][13] was valuated as insufficient. The model-based generative frameworks still seem to be challenged by the full expression of all required UIP aspects and are obliged to deliver a proof of concept by the evaluation of a set of representative UIPs.

Target architecture. Currently, there are no details available of the assumed architecture and integration of UIPs therein for the model-based generation. The task modeling and derivation of dialog structures often is focused by examples. In this regard, we wonder how complex *Dialog Logic* can be implemented, which demands for a number of branches due to user choices and results in different navigation options among UIP instances. Thus, it is not certain how closely task models and chosen UIPs for presentation of dialogs are coupled. In general, the complete configuration process of all related artifacts (tasks, dialogs, UIPs, application data and services) for the realization of a use case remains unknown. Lastly, it is uncertain whether manually implemented dialogs can be integrated among generated code or if every dialog specification results in the mandatory formalization of UIPs that may be used only once.

Tools. The generator based solutions require vast tool support for formalization, configuration or instantiation and finally transformations of UIPs. For the latter, two steps are necessary as UIPs and their parameters have to be transformed to a CUI model first, which is later used for final code generation or interpretation. There will be high efforts for maintaining the tool chain as well as related overhead for the definition of metamodels, rules and syntax validation. To integrate non UIP-based dialogs the developers will have to provide additional CUI specification facilities.

Platform. By using platform-independent models, the coupling of generation based solutions to certain infrastructures is generally low. Mostly, the paradigm of the GUI may be fixed to WIMP [2]. Thus, the UIP formalization is highly reusable. However, for each target platform suitable architectures and code templates have to be developed. Most parts of the generator code will be platform-specific transformations that are unlikely to be reused.

C. Virtual User Interface

Formalization of UIPs. In contrast to the generative approach, the VUI based solution does not necessarily depend on a separate notation for formalization. The formalization is realized by object-oriented CUI level code of the target platform programming language instead. We are inclined that an object-oriented language offers strong concepts that permit the vast flexibility of UIP expression. For instance, abstract classes with partly implemented operations may serve as ideal templates for UIP definitions.

 ISBN: 978-1-61208-367-4

The parameters for context adaptation can be set by operation parameters or separate setter operations. Furthermore, both structural and behavioral aspects can be combined in one specification unit. These basic facilities would have to be re-created by an external notation for a generation based solution. In this regard, even the template offering UIML 4.0 [24] GUI specification language lacks sufficient parameterization for UIPs [12] and would have to be extended.

In contrast, an OO programming language offers elementary functions to express any purpose or structure that may be improved by architectural or design patterns. Furthermore, the usage of an OO language for UIP expression is comparable to directly programming with a certain GUI framework to fulfill a certain domain's GUI requirements. Similar elementary facilities can be incorporated with the identified software categories for UIP expression, so that a high flexibility is achieved. The basic operations for presentation definition are based on the CUI level [29] and represent abstractions of common GUI framework facilities. They will both enable an accurate and abstract UIP formalization with a high flexibility due to the full range of OO language capabilities.

Initially, the UIP expression can be probed on the basis of the UIP analysis model [13]. The conceptual UIP modeling can be improved gradually without the need to adapt a specific notation and associated tools. With the basic foundations of factor [3] and analysis model [13], a rich information basis for UIPs is available that can be successively translated to code with the aid of the software category tree of Figure 3.

Target architecture. The VUI architecture is limited to presentation related tasks and does not include any assumptions concerning application integration. That means, each GUI design issue has to be solved from scratch or by the adaptation of available reference architectures. A solution tailored for UIP integration induces additional efforts but may result in an appropriate and reusable architecture.

Tools. The VUI needs no tools at all besides a compiler and an IDE that partly does the checking of programming language syntax. For visual impressions of defined UIPs, default configurations can be implemented, which may be used as test cases, too. The testing of UIP instances does not require additional inputs from external tools. The combination of UIPs and ordinary dialogs is possible without further adaptations.

Platform. For the VUI solution, the target platform language is fixed. There may be additional frameworks required, which permit the integration with different languages or even paradigms. But with each change of target language or GUI frameworks, the specific code for rendering has to be re-implemented. Therefore, the UIP formalization appears to be less reusable like the format used for the generation-based approach. But it may be ported to OO languages with comparable facilities, since the architecture is the key reusable artifact. In this regard, the architecture is based on interfaces and object-orientation, so that the VUI CUI components may partly be ported among different OO languages. Moreover, the formalization of UIPs is solely based on architecture components, interfaces and their interaction, so that no notation has to be adapted. In the end, the VUI solution may promise more reusable concepts, since they are not platform-specific like the transformations of a generator basis.

D. Outlook

The model-based generation approach raises many open issues concerning the UIP formalization and target architecture details. It is not certain when and what solutions are to appear. So, we opt for an alternative solution that is based on the VUI architecture.

V. Virtual User Interface Architecture Draft

As a result of the positive appraisal of the virtual user interface architecture, we will elaborate an architecture draft in the following paragraphs. The primary basis for the identification of components and their dependencies are provided by the software category models of Figure 2 and Figure 3. These categories need to be assigned to new components and their interfaces. The latter will clarify the dynamic behavior, which was not obviously described by the category trees. For the sake of keeping reference to the category trees, a similar naming of components was applied. In Figure 4, the structural architecture model is presented. Please note that not every software category will be represented as a component. The granularity of categories differs, so that some are assigned to components, classes (not visible here) or a set of operations modeled by interfaces.

A main component is modeled by the *Dialog*, which initiates application related behavior (*Dialog Logic*) and handles domain data (*Dialog Data Model*). Concerning the configuration of instances and initialization of visual components, the *Dialog Lifecycle Actions* are in charge.

Another main component is embodied by the *Presentation (FUI)* that serves as the final user interface with visual appearance and respective event handling. There exist two options for the instantiation of visual elements: Either simple UI-Controls can be initialized by the *UI-Control Configuration* or UIPs can be configured by *UIP Configuration*. Both components are associated to *Presentation Event Handling* to be able to have their elements linked to event processing. Triggers and state changes are decoupled by the separation of *Presentation Event Handling* and *View Definition*. The interfaces called by *View State Changes* represent operations that implement the results of visual state changes. When the received event is out of scope of the *Presentation (FUI)*, the *Event Forwarding* will call *Dialog Event Handling*. Moreover, the *Presentation Data Handling* is realized by the observer [31].

The *Virtual User Interface* component consists of one reusable (*Presentation (CUI)*) and one technical dependent (*Rendering*) component. As a consequence, there are always two representations of one UIP or UI-Control. The CUI level components of the *Presentation (CUI)* define the logical part of instances. In contrast, the *Rendering* creates corresponding technical instances that depend on the current *GUI Framework*. To decouple the CUI components from technical aspects, the *UIP Elements Definition* and *UIP Rendering* interfaces define the atomic operations required for both *UI-Control Definition* and *UIP Definition*.

 ISBN: 978-1-61208-367-4

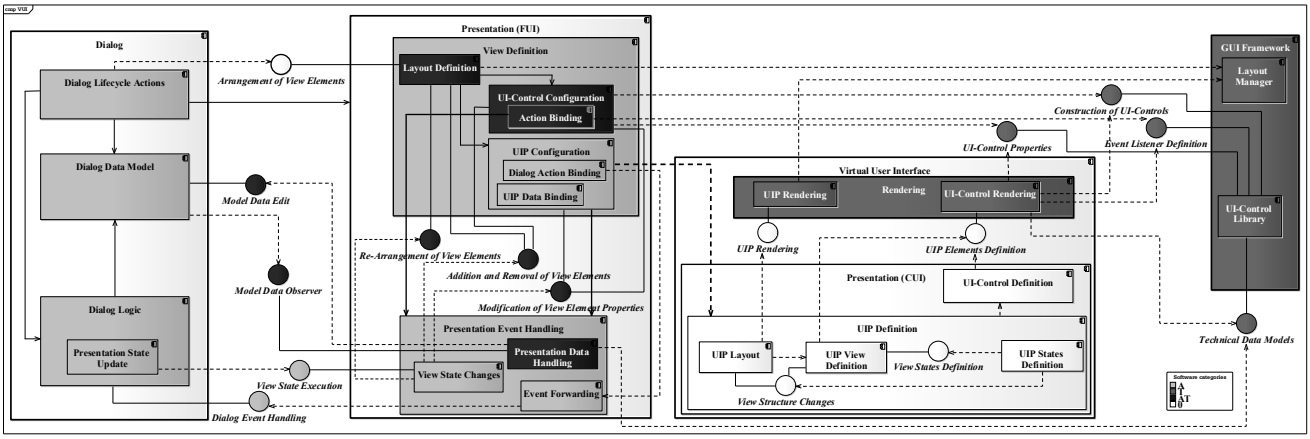


Figure 4. Virtual user interface architecture based on the UIP software categories.

These may be implemented by different *Rendering* components, which are specific for a certain *GUI Framework*. The versatile UIP formalization options are mostly assigned to *UIP Layout* and *UIP View Definition*. Depending on the current UIP instance configuration *UIP States Definition* may call the former components to trigger changes in visual or structural state.

VI. Results and Discussion

The vision to employ UIPs as reusable assets for a reduction of GUI implementation efforts cannot be realized by recent approaches because of the limited formalization of UIP aspects and variability. Besides, the general GUI design issues presented in Section II.B still persist due to the lack of detailed reference architectures and standardization. In Section III.B, we clearly pointed out how these issues impact the architecture for seamless UIP integration. A tight coupling to GUI frameworks can limit the UIP applicability. Also, important architecture concerns UIPs are connected with are without standardized solutions: navigation and application logic flow. Eventually, the integration of UIPs into GUI architectures has to overcome these issues.

Since UIP based solutions largely depend on reuse of basic GUI architecture concepts, UIP specific concerns have to be integrated and separated to reduce dependencies. In this context, we presented an enhanced software category model that addresses the prevailing GUI design issues and models typical UIP responsibilities. These categories can be used to identify a component based architecture for UIP implementation with separated concerns and limited dependencies. The identified *0* categories can be either generative sources or CUI level code of a VUI. In the end, the UIP category tree can also be helpful for generative development as it may identify aspects or components and separate them in order to enable a better maintenance of generator architectures.

As result of our comparison of general UIP implementation approaches, we opted for the unique VUI solution. The VUI solution promises a high flexibility of UIP formalization, platform independence and no additional tools or notation development efforts. On that basis, simple and complex UIPs can be relatively quickly probed for implementation. Please note that our analysis of mature XML GUI specification languages [12] revealed major limitations concerning UIP formalization that are hard to solve. UIP definitions may be better approached with OO language code.

Our VUI draft left the impression that much CUI based abstraction of common GUI framework concerns is required and that a complex architecture is anticipated. Representative UIPs have to be implemented to prove the VUI concept and refine its foundations. Due to UIP rendering needs of the VUI, the non UIP-based UI-Control compositions can benefit from the platform-independent rendering, too. In the end, the *AT* software character of *View Definition* components may be completely avoided.

The primary limitation of a VUI based solution will be its dependence on a strong OO language. One can argue that a VUI architecture is hard to establish for web-clients relying on browser based languages, such as JavaScript and popular frameworks like JQuery, due to lacking object-orientation. Frameworks like GWT [30] that are able to accept OO code and compile it to JavaScript may be a promising option for a VUI but can be limited due to the available set of UI-Controls. In the end, the CUI based code would need further enhancements to represent alternative definitions of UIPs currently not covered by present UI-Controls.

Finally, a VUI based approach will not be achieved without obstacles. The abstraction of common GUI framework operations to CUI level code for reuse by UIP definitions is not an easy task. Moreover, the design of interfaces and their operations has to suit current and future UIP definitions. The software category tree will help us to limit framework dependencies and plan the distribution of responsibilities among components.

VII. Conclusion and Future Work

In the future, UIPs are likely to become complementary assets for reuse in comparison to design patterns [31]. With the incorporation of UIPs as valuable assets for the reuse of parts of the implementation code, the complexity of GUI artifacts to be designed and developed manually would be reduced. Much of the former GUI programming would be replaced by configuration of chosen UIP instances. As a consequence, the developers could focus more on application relevant design. However, current approaches that employ

 ISBN: 978-1-61208-367-4

UIPs on the basis of model-based generation are still challenged by formalization issues and have not proven their UIP variability concepts yet.

Future work. The alternative VUI based approach will be further elaborated in our future work. At first, the common GUI design issues have to be solved by a detailed GUI reference architecture. On the basis of the presented software category models and our VUI draft, we will be able to identify a suitable component based architecture. The requirements for a VUI based solution will be complemented by example UIPs and implementations. During that process, both category and UIP requirements models will be updated. Finally, we will investigate on the impacts of UIPs on other architecture artifacts and their traceability connections.

REFERENCES

[1] J. Tidwell, Designing Interfaces. Patterns for Effective Interaction Design. Beijing: O'Reilly, 2006.

[2] D. Ammon, S. Wendler, T. Kikova, and I. Philippow, "Specification of formalized software patterns for the development of user interfaces," The Seventh International Conference on Software Engineering Advances (ICSEA 12) IARIA, Nov. 2012, pp. 296-303, ISBN: 978-1-61208-230-1.

[3] S. Wendler, D. Ammon, I. Philippow, and D. Streitferdt "A factor model capturing requirements for generative user interface patterns," The Fifth International Conferences on Pervasive Patterns and Applications (PATTERNS 13), IARIA, May 27 - June 1 2013, pp. 34-43, ISSN: 2308-3557.

[4] F. Radeke and P. Forbrig, "Patterns in task-based modeling of user interfaces," The Sixth International Workshop on Task Models and Diagrams for Users Interface Design (TAMODIA 07), Springer LNCS 4849, Nov. 2007, pp. 184-197.

[5] J. Engel and C. Märtin, "PaMGIS: A framework for pattern-based modeling and generation of interactive systems," The Thirteenth International Conference on Human-Computer Interaction (HCII 09), Part I, Springer LNCS 5610, July 2009, pp. 826-835.

[6] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stahl, Pattern-Oriented Software Architecture, Volume 1: A System of Patterns. New York: John Wiley & Sons, 1996.

[7] M. Fowler, Patterns of Enterprise Application Architecture. New Jersey: Addison-Wesley Professional, 2003.

[8] M. Fowler, "Development of Further Patterns of Enterprise Application Architecture," http://martinfowler.com/eaaDev/index.html, 2014.08.14.

[9] M. Haft, B. Humm, and J. Siedersleben, "The architect's dilemma – will reference architectures help?," First International Conference on the Quality of Software Architectures (QoSA 2005), Springer LNCS 3712, Sept. 2005, pp. 106-122.

[10] S. Wendler, "A software category model for graphical user interfaces," The Ninth International Conference on Software Engineering Advances (ICSEA 2014), IARIA, in press.

[11] S. Siedersleben, Ed., Softwaretechnik: Praxiswissen für Software-Ingenieure [Software engineering: practical knowledge for software engineers], 2nd ed. München: Hanser, 2003.

[12] S. Wendler, D. Ammon, T. Kikova, I. Philippow, and D. Streitferdt, "Theoretical and practical implications of user interface patterns applied for the development of graphical user interfaces," International Journal on Advances in Software, vol. 6, nr. 1 & 2, pp. 25-44, 2013, IARIA, ISSN: 1942-2628, http://www.iariajournals.org/software/.

[13] S. Wendler and D. Streitferdt, "An analysis of the generative user interface pattern structure," International Journal On Advances in Intelligent Systems, vol. 7, nr. 1 & 2, pp. 113-134, 2014, IARIA, ISSN: 1942-2679, http://www.iariajournals.org/intelligent_systems/index.html.

[14] J. Siedersleben and E. Denert, "Wie baut man Informationssysteme? Überlegungen zur Standardarchitektur [How to build information systems? Thoughts on a standard architecture]," Informatik Spektrum, vol. 23, issue 4, Aug. 2000, pp. 247-257, doi: 10.1007/s002870000110.

[15] J. Siedersleben, Moderne Softwarearchitektur [Modern software architecture], 1st ed. 2004, corrected reprint. Heidelberg: dpunkt, 2006.

[16] S. Alpaev, "Applied MVC patterns. A pattern language," The Computing Research Repository (CoRR), May 2006, http://arxiv.org/abs/cs/0605020, 2014.08.14.

[17] A. Karagkasidis, "Developing GUI applications: architectural patterns revisited," The Thirteenth Annual European Conference on Pattern Languages of Programming (EuroPLoP 2008), CEUR-WS.org, July 2008.

[18] A. Dearden and J. Finlay, "Pattern languages in HCI; A critical review," Human-Computer Interaction, vol. 21, issue 1, 2006, pp. 49-102.

[19] S. Wendler and I. Philippow, "Requirements for a definition of generative user interface patterns," The Fifteenth International Conference on Human-Computer Interaction (HCII 13), Part I, Springer LNCS 8004, July 2013, pp. 510-520.

[20] A. Wolff, P. Forbrig, A. Dittmar, and D. Reichart, "Tool support for an evolutionary design process using patterns," Workshop on Multi-channel Adaptive Context-sensitive Systems (MAC 06), May 2006, pp. 71-80.

[21] UsiXML website, http://www.usixml.org/, 2014.08.14.

[22] K. Breiner, K. Bizik, T. Rauch, M. Seissler, G. Meixner, and P. Diebold, "Automatic adaptation of user workflows within model-based user interface generation during runtime on the example of the smartmote," The Fourteenth International Conference on Human-Computer Interaction. Design and Development Approaches. (HCII 2011), Part I, Springer LNCS 6761, July 2011, pp. 165-174.

[23] K. Breiner, G. Meixner, D. Rombach, M. Seissler, and D. Zühlke, "Efficient generation of ambient intelligent user interfaces," The Fifteenth International Conference on Knowledge-Based and Intelligent Information and Engineering Systems (KES 11), Springer LNCS 6884, Sept. 2011, pp. 136-145.

[24] UIML 4.0 specification, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uiml, 2014.08.14.

[25] G. Meixner, M. Seissler, and K. Breiner, "Model-Driven useware engineering," Model-Driven Development of Advanced User Interfaces, Studies in Computational Intelligence, vol. 340, H. Hussmann, G. Meixner, and D. Zuehlke, Eds., Berlin, Heidelberg: Springer, pp. 1-26.

[26] K. Bittner and I. Spence, Use case modeling, 8th print. Boston (Mass.): Addison-Wesley, 2006.

[27] M. Seissler, K. Breiner, and G. Meixner, "Towards Pattern-Driven Engineering of Run-Time Adaptive User Interfaces for Smart Production Environments," The Fourteenth International Conference on Human-Computer Interaction (HCII 11), Springer LNCS 6761, July 2011, pp. 299-308.

[28] M. van Welie, "A pattern library for interaction design," http://www.welie.com, 2014.08.14.

[29] J. Vanderdonckt, "A MDA-compliant environment for developing user interfaces of information systems," The Seventeenth International Conference on Advanced Information Systems Engineering (CAiSE 2005), Springer LNCS 3520, June 2005, pp. 16-31.

[30] Google Web Toolkit, http://www.gwtproject.org/, 2014.08.14.

[31] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-oriented Software. Reading: Addison-Wesley, 1995.

 ISBN: 978-1-61208-367-4

A Guideline for Supporting Agile Process Assessments

Teresa M. M. Maciel
Department of Statistic and Informatics
Federal Rural University of Pernambuco
Rua Dom Manoel de Medeiros, s/n,
Recife, Brazil
tmmaciel@gmail.com

Silvio R. L. Meira
Informatics Center
Federal University of Pernambuco
Av. Jornalista Anibal Fernandes, s/n
Recife, Brazil
srlm@cin.ufpe.br

***Abstract*—A critical factor in determining whether a company achieves competitive advantage in the market is its ability to deal with unexpected and continuous changes. This critical determinant is addressed by the term "agility". The current paper proposes a methodology for assessing agility at the organizational level, based around a reference model governed by a set of agile capabilities. The capabilities were selected from a review of the relevant literature in the manufacturing and software development fields. Along with this capability set, the reference model identifies an array of enablers and metrics, which facilitate their implementation. Finally, a case study discusses the experience of applying the proposal in the real environment of an established software company.**

Keywords: software agility evaluation; agile assessment.

I. INTRODUCTION

A critical factor in determining whether a company achieves competitive advantage in the market is its ability to deal with unexpected and continuous changes. This critical determinant is addressed by the term "agility". Companies must reconfigure all the various elements of which they are composed (human, managerial, and technological) in order to successfully adopt agile methodologies.

Both the manufacturing and software development fields have faced similar challenges in recent years. Indeed, due to the commonalities between the fields some authors assign the core ideas in agile software development to trends in the manufacturing area [4][6][23].

The concept of agility was first formalized in a report entitiled '21st Century Manufacturing Enterprise Strategy', published by the Iacocca Institute in 1991 [3][11][10][15]. In this work agility is defined as a strategic ability, suggesting that being agile means being proficient at change. Consequently, a number of works were published in the literature which focused on refining the meaning of the term [14][8][10][20][21]. In software development context, new proposals emerged in the 90s as Scrum [18], XP [1], Crystal Clear [2], FDD [13], and DSDM [19].

Some authors promote ways for assessing agility, as in [17]. However, few are concerned with assessing agility from software organizations perspective.

This paper presents a model to support software organizations assess their agility status. Section II shows a brief overview about agility evaluation. Section III describes a reference model proposed to serve as a basis to the assessment process. Section IV reports a study applied in a real organization. Finally, Section V presents the conclusions of this work.

II. AGILITY EVALUATION

Several efforts have been published in order to propose ways to evaluate organizational agility. Sharifi and Zhang [28][29] proposed a conceptual model with agility drivers, capabilities and providers.

Meredith and Francis [6] defined a set of agility components organized into four categories: agile strategies, agile processes, agile linkages, and agile people. Maskell [12] defined four elements of Agile Manufacturing: customer prosperity; people and information; cooperation; and fitness for change. Jin-Hai et al. [10] proposed a concept they called "real agile manufacturing" based on the critical aspects of strategic processing, multiple winners, integration, core competence, and IT. Ramesh [15] presented a literature review by identifying a set of criteria for attaining agility and also suggested a procedure for its successful implementation. Dove [4][27] stated that "Being Agile means being proficient at change – and allows an organization to do anything it wants to do whenever it wants to do". Plonka [14] specifies the critical attributes of an agile workforce: an attitude towards learning and self-development; problem-solving ability; being comfortable with change, new ideas and technologies; the ability to generate innovative ideas; along with the readiness to accept new responsibilities. Gunasekaran [8] defined a set of characteristics for agile teams: self-directed, containing IT-skilled workers with knowledge of team working, negotiation, advanced manufacturing strategies and technologies, who are also empowered, multifunctional and multilingual.

The assessment process proposed by this work is set within the context of software development at an organizational level, and comprises the reference model, assessment team, the company, and the evaluation process. Figure 1 illustrates the assessment environment.

 ISBN: 978-1-61208-367-4

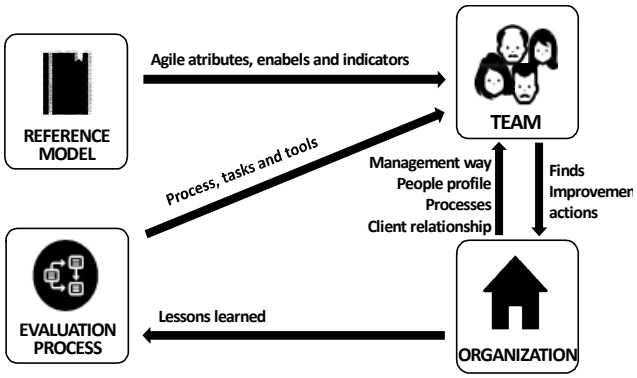


Figure 1. Assessment Environment

III. THE REFERENCE MODEL

As shown in Figure 1, the team assesses the software company guided by the reference model and the evaluation process, making discoveries and recommending actions to increase agility levels within the company. Figure 2 illustrates the model in detail and lists the specific agile attributes, enablers, and indicators on which the assessment process is based.

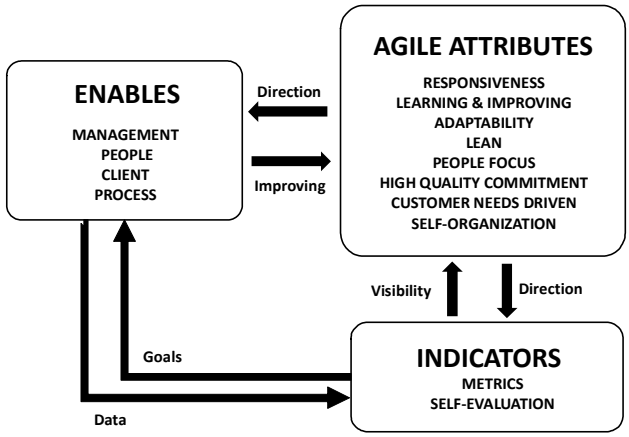


Figure 2. The Reference Model

As listed in the figure, the set of agile attributes includes the following characteristics: responsiveness, learning and improvement oriented; adaptability; lean; focus on people; commitment to high quality; driven by customer needs; and self-organization. It is this set of attributes that propels the enablers to implement and improve the agile capabilities of the company. At the same time, they establish the indicators, which are first obtained from metrics or evaluation results and then executed by accepted practices and tools.

A. Software Agility Capabilities

To identify the common capabilities and attributes that define an agile company, a literature review was conducted in both the manufacturing and software fields. The review considered works that contained the following keywords; 'agile attribute', 'agile criterion', 'agile concept', 'agile definition', or 'agile capabilities'. The search process focused on articles from the following sources: ACM Digital Library; IEEE Computer Society Digital Library; and Google Scholar (in order to widen the search). The results of the search, the majority of which came the manufacturing research field, were combined to construct Table I, which shows that agility converges into six common capabilities.

1) Responsive

The quality of responsiveness can be defined as the capability to easily accept and deal with changes; to identify changes and respond to them both reactively and proactively; and to recover from them [28].

2) Fast

Since agility is a rapid and proactive adaptation to continuous and unpredicted changes, speed is an essential attribute of an agile organization. A fast organization gets to the market quickly, with a production time that guarantees the fast delivery of products and services. However, this capability should not be limited to the time of production: it must be evident throughout the company. Some authors, Breu et al. [21] for example, cite the speeds of skill development, adaptation to new work environments, and information access as indicators for evaluating the agility of a workforce.

TABLE I. SOFTWARE AGILE ATTRIBUTES

Agile Capability	Author
Responsive	[1]; [3]; [4]; [9]; [12]; [18]; [21]; [23]; [24]; [26]; [28]; [30].
Fast	[1]; [3]; [4]; [7]; [9]; [18]; [21]; [24]; [27]; [28]; [30].
Adaptable	[1]; [3]; [4]; [18]; [23]; [24]; [27]; [28]; [30].
Knowledge-driven	[7]; [18]; [21]; [23]; [24]; [25]; [28]; [30].
Self-organized	[7]; [9]; [12]; [18]; [23]; [24]; [28]; [30].
Quality and Improving Committed	[1]; [7]; [9]; [18]; [23]; [24]; [30].

 ISBN: 978-1-61208-367-4

3) Adaptable

Adaptability is commonly related to flexibility. To adapt itself to the demands of the market, a company requires flexible processes and structures, as well as flexible people. The concept of organizational adaptability originated from the contingency approach in organizational research, and the theory that the organizing style is dependent on the situational constraints of the environment in which the company operates [30].

4) Knowledge-Driven

According to the literature review, a focus on knowledge represents a critical aspect of agility. Goldman et al. [25] for example, link competitive agile environments to knowledge and experience. Yusuf et al [23] cite that the best practices of a knowledge-rich environment provide the means to produce customer-driven products in a fast changing environment. Sherehiy [30] states decentralized knowledge as a characteristic of an organic organizational design. Knowledge management and change proficiency are co-dependent relationships, and the enabling competencies of an agile company [5]. The emphasis on short development cycles, reviews, collaborative work and retrospectives found in the Agile Manifesto, also agile practices such as XP and Scrum, reflect the importance given to organizational knowledge in software development.

5) Self-organized

Organizational agility demands proactive and adaptive responses, and thus certain key skills are required in an agile workforce. Based on the evidence from the review, these skills are directly linked to empowered and self-organized teams. Particularly considering the environment of agile software development, the requirement for a high level of both individual and team autonomy is viewed as a prerequisite attribute.The Agile Manifesto explicitly includes this aspect in one of its 12 agile principles, affirming that the best architectures, requirements, and designs emerge from self-organizing teams.

6) Quality and Improving Committed

As agility is a dynamic and competitive ability, its institutionalization demands high quality with decreasing lead time [4]. Retrospectives are recommended by agile methods as a mechanism to discover means to increase effectiveness. According to the Agile Manifesto, a team tunes and adjusts its behavior driven by the desire to improve its performance.

B. Supporting Enablers

Agile capabilities represent the direction of organizational improvement, and are achieved through the implementation of accepted practices and tools (enablers). Institutionalized agile enablers are the clear indicators of the agility journey during an agile assessment evaluation, therefore agile companies must understand and identify which enablers are appropriate for each project or program. Table II presents a set of common enablers selected from the results of the literature review and connects each capability with its relevant agile capability as stated by the Agile Alliance in its Agile Practices Guide [31].

TABLE II. AGILE CAPABILITIES AND SUPPORTING ENABLERS

Capabilities	**Enablers**
Responsive	Continuous deployment, Frequent releases, Daily meeting, Incremental development, Rules of simplicity
Fast	Automated build, Automated test, Continuous delivery, Continuous integration, Incremental development, Planning Poker, Rules of simplicity
Adaptable	Automated test, Continuous integration, Daily meeting, Frequent releases, Incremental development, Pair programming, Rules of simplicity
Knowledge-driven	Pair programming, Retrospectives, reviews, Collective ownership, Incremental development, Kanban boards, Refactoring
Self-organized	Daily meeting, Retrospective, Kanban boards, Planning Poker
Quality and Improving Committed	Acceptance test, TDD, Daily meeting, Retrospective, Incremental development, INVEST, Kanban boards, Pair programming, Refactoring, Usability tests

C. Supporting Metrics

In order to evaluate the improvement of agility, the assessment guideline considers a set of metrics to be monitoring. Table III shows the list of metrics adopted. Besides, the table presents the mapping between each capability and the supporting metrics.

Table III helps software organizations to identify what to consider to measure in order to monitoring its agile way. Metrics listed should be collected and tracking at the organizational level, and it can be a team, a project, a department or the whole company.

Some of the metrics support directly one specific capability. For example, Lead Time (the time between the initiation and completion of a production process) is related directly to capability fast. Throughput, on the other side, is related to self-organized indirectly.

The evolution of each metric should be closely monitored and analyzed during the assessment to identify both the capabilities that are improving and the key areas that require further work.

 ISBN: 978-1-61208-367-4

TABLE III. AGILE CAPABILITIES AND SUPPORTING METRICS

METRIC	CAPABILITIES: RESPONSIVE	FAST	ADAPTABLE	KNOWLEDGE-DRIVEN	SELF-ORGANIZED	QUALITY AND IMPROVEMENT COMMITED
Cost of change (effort)	X	X	X		X	
Time to change	X	X	X		X	
Improvement frequency			X	X		X
Lead Time	X	X				
Throughput	X	X			X	
Takt time	X	X				
Team building speed	X	X	X		X	
Client Satisfaction		X				X
Requirements or BV burn downs	X	X		X		X
Role variety	X		X	X	X	
Cumulative Flow		X		X		
Re-work measurement				X		X
Technical Debt	X					X
Defect Density	X					X

IV. ASSESSMENT PROCESS APPLIED IN A REAL CASE STUDY

The goal of the case study was to validate the suitability of the proposal in a real situation. The assessment process was applied to a small Brazilian company that produces software in the industrial automation area and first adopted agile and Lean approaches in 2011. At the beginning of the process, the complete team consisted of 20 employees, of whom 15 were directly involved in software development and the remainder in administration, marketing and sales. During the assessment period of 3 months a set of metrics was collected at monthly intervals to verify the degree of improvement in agile capabilities.

An analysis of the data available in the company was performed to define a set of suitable metrics. These were: lead time (the period of time between the beginning and the end of user story development); throughput (number of user stories divided by total time); takt time (the total time divided by the number of user stories); improvement frequency (the number of actions implemented as a result of retrospectives); and client satisfaction (collected from a systematic monthly survey made by the organization). Figure 3 is a selection of metrics for one small project over the three month period.

According to Figure 3, throughput, takt time and lead time all demonstrated improving curves during the study. 59 user stories (attributed by the team as smalls) were collected and developed. In the first month, the team produced a thoughput of 0.5 user story, improving to 0.7 and 0.8 in the consecutive months. Takt time values demonstrated a similar improving behavior. This represents a good indicator of capabilities as being responsive and quality and improvement committed. Similar results were found in the other collected metrics. It is important to state that during this period the organization applied enablers in order to improve its results.

Each metric was analyzed in terms of its institutionalization as well as its application for improvement. Parallel observation and self-evaluation was performed by the team to verify the level of institutionalization of each agile metric or practice. Table IV gives the results of the evaluation, where each agile enabler or metric was evaluated by the team as Institutionalized (I), In Progress (P), or Not Worked (N).

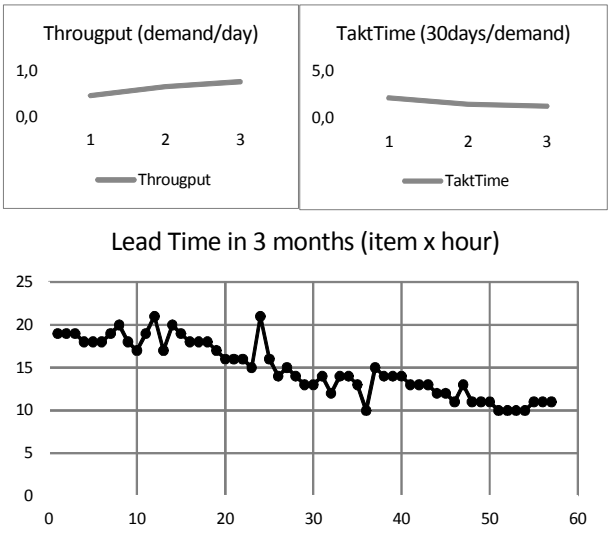


Figure 3. Throughput, Lead Time and Takt Time collected from case study

Figures 4 and 5 give a graphical view of the evaluation results in terms of the degree of institutionalization for each agile practice and the attendance of a capability.

The case study results highlight areas of improvement, demonstrating the impact of these improvements in terms of agile capabilities. It should be stated that it is not mandatory for a company to implement all the recommended agile practices. That is the reason that an agile capability being worked through at least one enabler and metric was classified as 'In Progress' in the evaluation.

 ISBN: 978-1-61208-367-4

TABLE IV. AGILE CAPABILITIES ATTENDANCE ANALYSIS

	CAPABILITIES					
METRIC	RESPONSIVE	FAST	ADAPTABLE	KNOWLEDGE-DRIVEN	SELF-ORGANIZED	QUALITY AND IMPROVEMENT COMMITED
Cost of change (effort)	N	N	N		N	
Time to change	N	N	N		N	
Improvement frequency			P	P		P
Lead Time	P	P				
Throughput	P	P			P	
Takt time	P	P				
Team building speed	N	N	N		N	
Client Satisfaction		I				I
Requirements or BV burn downs	P	P		P		P
Role variety	N		N	N	N	
Cumulative Flow		P		P		
Re-work measurement				N		N
Technical Debt	N					N
Defect Density	P					P
How we monitoring the agile capabilities attendance?	P	P	P	P	P	P
ENABLER						
Continuous deployment	P	P	P			
Frequent release	I	P	I			
Continuous delivery	I	I	I			
Daily meeting		I	I	I	I	I
Incremental development	I		I	I		
Rules of simplicity	N	I	N			
Pair programming		N	N	N	N	N
Retrospectives, reviews				P	P	P
Collective ownership			P	P	P	
Kanban boards				I	I	I
Refactoring				N		N
Automated test	P		P			P
Continuous integration	N	P				N
Acceptance test	I	N				I
TDD	N					N
INVEST	N					N
Planning Poker	P				P	P
Automated build	P	P	P			
Usability tests		P	N			N
How we apply enablers to intensify agile capabilities attendance?	P	N	P	P	P	P

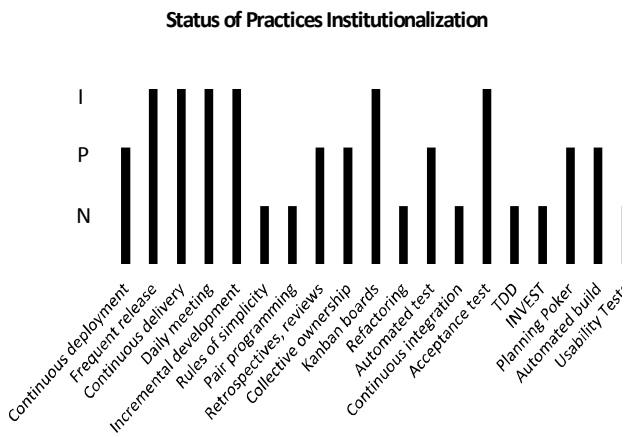


Figure 4. Agile Practices Institutionalization

V. CONCLUSION

A proposal was presented for assessing organizational agility which could be applied at company, department, and team levels. The basis of the proposal is a reference model that is driven by a set of agile capabilities selected from a literature review carried out in the manufacturing and software development fields. To accompany this set of capabilities, the model offers an array of enablers and metrics that can facilitate a company to achieve these agile capabilities. Each agile enabler was linked to the capability it supported and demonstrated the relationship between each capability and its supporting metrics. Finally, a case study was included to illustrate the experience of the proposal being applied to a small Brazilian software company.

ACKNOWLEDGMENT

This work was (partially) supported by the National Institute of Science and Technology for Software Engineering (INES[1]), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08.

REFERENCES

[1] K. Beck, “Extreme Programming Explained: Embrace Change”, Addison-Wesley, ISBN 0-201-61641-6, 2000.

[2] A. Cockburn, “Crystal Clear: A Human-Powered Methodology for Small Teams”, Addison-Wesley, ISBN 0-201-69947-8, 2004.

[3] R. Dove and R.N. Nagel, “21st Century Manufacturing Enterprise Strategy: An Industry Led View”. Iacocca Institute, Lehigh University, Bethlehem, PA, 1991.

[4] R. Dove, “Lean And Agile: Synergy, Contrast, And Emerging Structure”. Proceedings of the Defense Manufacturing Conference’93, San Francisco, CA, 1993.

[1] www.ines.org.br

[5] R. Dove, “Response Ability: the Language, Structure, and Culture of the Agile Enterprise. Wiley, New York, , 2001.

[6] S. Meredith, D. Francis, "Journey towards agility: the agile wheel explored", The TQM Magazine , 2001, vol.12 iss: 2, pp.137 – 143

[7] M. Fowler and M. Highsmith, “The Agile Manifesto,” Software Development, Aug. 2001, pp. 28-32.

[8] A. Gunasekaran, “Agile Manufacturing: A Framework for Research and Development”. International Journal of Production Economics, 1999, vol.62, pp.87–105.

[9] J. Highsmith, A. Cockburn, “Agile Software Development: The People Factor”, IEEE Computer Magazine, 2001, pp. 133-136.

[10] L. Jin-Hai, A. R. Anderson, and R. T. Harrison, “The evolution of agile manufacturing”, Business Process Management Journal, 2003, vol.9 No. 2, pp. 170-89.

[11] S. Izza, R. Imache, L. Vincent, and Y. Lounis, “An Approach for the Evaluation of the Agility in the Context of Enterprise Interoperability”, Enterprise Interoperability III, Springer, ISBN 978-1-84800-220-3, 2008.

[12] B. Maskell, “The age of agile manufacturing”, Supply Chain Management: An International Journal, 2001, vol.6 No. 1, pp. 5-11.

[13] S. R. Palmer, Felsing, J.M., “A Practical Guide to Feature-driven Development”, Prentice Hall, Upper Saddle River, NJ, ISBN 0-13-067615-2, 2002.

[14] F. S. Plonka, “Developing a lean and agile work force”. Human Factors and Ergonomics in Manufacturing, 1997, vol.7 No.1, pp.11–20.

[15] C. Ramesh, “Literature review on the agile manufacturing criteria”, Journal of Manufacturing Technology Management, 2007, vol.18 No. 2, pp. 182-201.

[16] D. J. Reiffer, “How Good are Agile Methods?”, IEEE Software, 14–17, 2002.

[17] B. Rumpe and A. Schroder, “Quantitative survey on extreme programming projects”, Proceedings of International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2002) Alghero, Italy, (2002, pp. 95–100.

[18] K. Schwaber, M. Beedle, “Agile Software Development with Scrum”, Prentice Hall, Upper Saddle River, 2001.

[19] J. Stapleton, “DSDM: Business Focused Development”, Second Ed., Pearson Education, ISBN 978-0321112248.Version 2.3, Keele University and University of Durham, EBSE Technical Report, 2007.

[20] J. Sarkis, “Benchmarking for agility”, Benchmarking: An International Journal, MCB University Press, vol.8 No. 2, 2001, pp. 88-107.

[21] K. Breu, C. Hemingway, M. Strathern, “Workforce agility: The new employee strategy for the knowledge economy”. J. Information Technology, 2001, pp. 17 21–31.

[22] P. Kentunen, “Adopting key lessons from agile manufacturing to agile software product development—A comparative study”, Technovation, 2008, vol.29 (2009) 408–422.

[23] Y. Yusuf, M. Sarhadi, A. Gunasekaran, “Agile manufacturing: the drivers, concepts and attributes”. International Journal of Production Economics, 1999, vol.62, pp. 33–43.

[24] P. T. Kidd, “Agile Manufacturing: Forging New Frontiers”. Addison-Wesley Reading, 1994.

[25] S. L. Goldman, R. N. Nagel, and K. Preiss, “Agile Competitors and Virtual Organizations: Strategies for Enriching the Customer”, Van Nostrand Reinhold, 1995.

[26] A. Nelson and F. A. Harvey, “Technologies for Training and Supporting Your Agile Workforce”, Proceedings 4th Agility Forum Annual Conference, Bethlehem, PA, 1995.

[27] R. Dove, “The Meaning of Life and the Meaning of Agile”, Production Magazine, 1994.

[28] H. Sharifi and Z. Zhang, “A Methodology For Achieving Agility In Manufacturing Organisations: An Introduction”, International Journal of Production Economics, 1999, vol.62, pp. 7–22.

[29] Z. Zhang and H. Sharifi, “A Methodology for Achieving Agility in Manufacturing Organisations”, International Journal of Operations & Production Management, 2000, vol.20, pp. 496-512.

[30] B. Sherehiy, K. Karwowski, J. K. Layer, “A Review of Enterprise Agility: Concepts, Frameworks, and Attributes, International Journal of Industrial Ergonomics”, 2007, vol.37, pp. 445–460.

[31] Agile Alliance, “Agile Practices Guide”, accessible through http://guide.agilealliance.org/.

 ISBN: 978-1-61208-367-4

Usage of Kanban in Software Companies

An empirical study on motivation, benefits and challenges

Muhammad Ovais Ahmad, Jouni Markkula, Markku Oivo, Pasi Kuvaja
Department of Information Processing Science
University of Oulu
Finland
e-mail: {Muhammad. Ahmad, Jouni. Markkula, Markku. Oivo, Pasi. Kuvaja}@oulu.fi

Abstract— **There is a growing interest in applying Kanban in software development to reap the proclaimed benefits presented in the literature. The goal of this paper is to provide up-to-date knowledge of the current state of Kanban usage in software companies, regarding the motivation for using it as well as the benefits obtained and challenges faced in its adoption. In addition, we investigate how the challenges identified in the study can be addressed. For this purpose, an empirical study was conducted consisting of a survey and complementing thematic interviews. The empirical study was carried out in November-December 2013 within large Finnish software companies, which extensively use Agile and Lean approaches. The obtained results are largely in line with the findings of earlier research reported in the literature. Generally, the experiences of using Kanban are rather positive; however, challenges in adoption identified include a lack of specialised training and usage experience, and a too traditional organisational culture.**

Keywords- ***Kanban; Lean; Agile; Software development.***

I. INTRODUCTION

Today, Agile and Lean are used in most global industries including the software industry. The Lean approach has been developed and found successful in the manufacturing industry [1], and it was later adopted by the software engineering field.

The key trait of Lean is to eliminate all kinds of waste from the development. Poppendieck and Poppendieck [1] transformed Lean manufacturing principles for use in the software engineering field. In order to eliminate and manage waste Poppendieck and Poppendieck [1] proposed the following set of principles:

- Build quality in
- Create knowledge
- Defer commitment
- Deliver fast
- Respect people
- Optimise the whole

In software development, all things that do not produce value for the customer are considered "waste". For example, partially completed work, extra processes, extra features, task switching and defects are also considered a waste [1].

Kanban is one way to execute Lean thinking, used for decades in managing production operations at Toyota [2]. It is the most recent addition to the Agile and Lean software development. In recent years, Kanban has become more popular in software development. A strong practitioner-driven movement emerged to supporting it [3][4]. Currently, Kanban method is being increasingly adopted to complement Scrum and other Agile methods in software processes.

Despite the recent increasing interest in Kanban among practitioners, existing scientific literature addresses it infrequently in the context of software development. Only a few studies on Kanban usage, how it is carried out in practice and its effects in software development have been published [5]. With the goal of providing up-to-date results that can be utilised by organisations implementing or planning to implement Agile and Lean methods, we have conducted an empirical study on the current state of Kanban usage in software companies. We carried out the study in the Cloud software and Need for Speed program, which is a large Finnish national program aiming at improving the competitive position of the Finnish software intensive industry in global markets by pioneering the building of new cloud business models, a Lean software enterprise model and open cloud software infrastructure. The program involves more than 30 research organisations and enterprises, including most of the major Finnish software companies that are actively using and researching Agile and Lean [5]. We conducted a survey and a set of interviews addressing the following questions:

- Why is Kanban used in software companies?
- What are the benefits, of using Kanban?
- What are the challenges in adopting Kanban and its solutions?

The rest of the paper is structured as follows: Section II reviews the current literature on Kanban usage. Section III describes the research setting, including the process for collecting the empirical data as well as the design of the survey and interviews. Section IV present the results of the study and compares them with the findings from earlier research. Finally, Section V concludes the paper summarising the results and discussing the limitations of the study.

 ISBN: 978-1-61208-367-4

II. KANBAN BACKGROUND

Kanban, developed by Taiichi Ohno, was introduced in the Japanese manufacturing industry in the 1950s. Kanban literally means signboard or visualisation of inventory used in the scheduling system for just in time (JIT) production. The idea behind the development of Kanban was to find a system to maintain and improve production. It is a flow control mechanism for pull-driven JIT production, in which the upstream processing activities are triggered by the downstream process demand signals [2][7]. Kanban runs the production system as a whole, and it has proven an excellent way of promoting improvement. It was successfully used in practice by Toyota. The basic idea behind Kanban usage is to execute Lean thinking in practice; however Lean is more than Kanban [7][8][9].

Present-day understanding of Lean and Kanban in software development is largely driven by practitioner's books [1][6][10]. The Lean and Kanban principles appear to be largely overlapping, reflecting the same grounding. Table I shows the Lean software development principles [1] and Kanban principles [10], which are known to the Agile community.

TABLE I. LEAN AND KANBAN PRINCIPLES

Lean software development principles [1]	Kanban Principles [9]
Eliminate waste Build quality in Create knowledge Defer commitment Deliver fast Respect people Optimise the whole	Visualise the workflow Limit work in progress Measure and manage flow Make process policies explicit Improve collaboratively

Kanban in software development originated in 2004, when David J. Anderson [10] was assisting a small IT team at Microsoft that was operating poorly. Anderson [10] introduced Kanban to the team to help the team members visualise their work and put limits on their work in progress (WIP). The motivation behind visualisation and limiting WIP was to identify the constraints of the process and focus on a single item at a time. This technique promotes the pull approach. In traditional software development, the work items are given i.e. "pushed" to each team member, who are then instructed to finish as many of them as quickly as possible. The traditional development work is in the form of a chain in which one team member's work item is handed over to another i.e. from developer to tester. This causes delays in the whole process when the next member in the line is overloaded or has a problem with his/her work. Kanban works in an alternative way. Instead of pushing work items, it promotes a pull system. Each member of a team has one item to work on at a time. When he/she finishes it, he/she will automatically pull the next item to work on.

In brief, Kanban aims to provide visibility to the software development process, communicate priorities and highlight bottlenecks [5], which results in a constant flow of releasing work items to the customers, as the developers focus only on those few items at a given time [6]. Figure 1 shows the typical structure of a Kanban board and its principles in action.

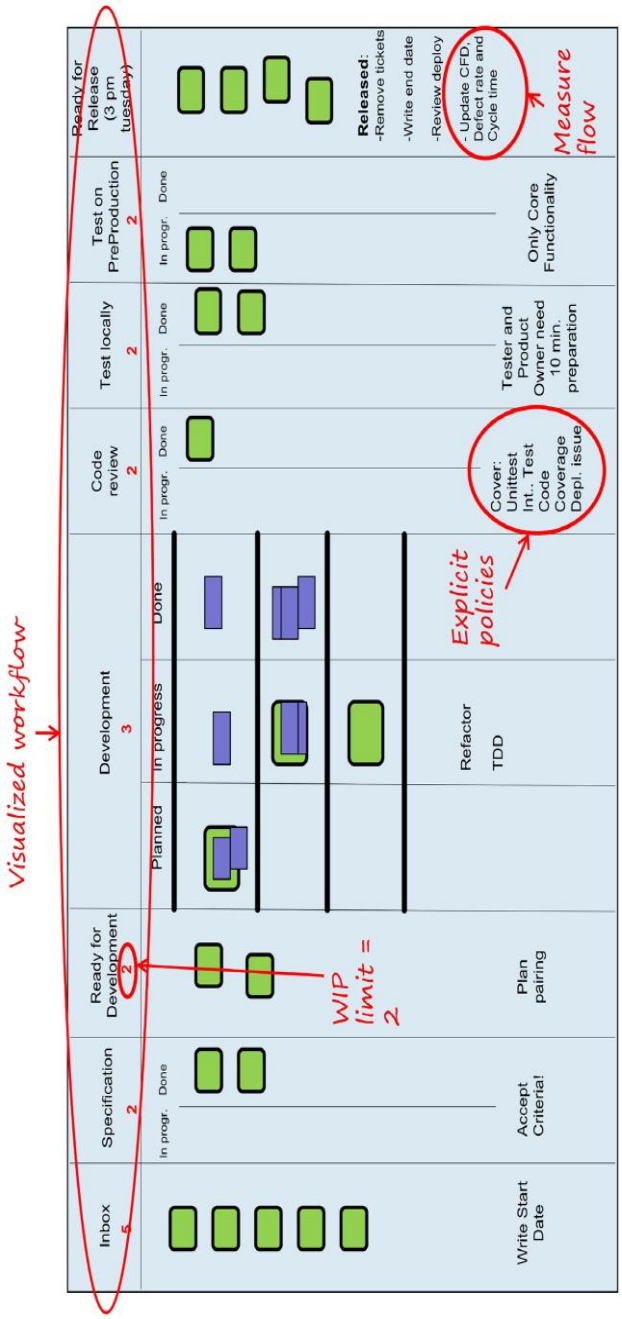


Figure 1. Kanban board and principles in action [12]

The Kanban method in software development drives project teams to visualise the workflow, limit WIP at each workflow stage, and measure the cycle time [11]. The key motivation for the usage of Kanban is to focus on flow and the absence of obligatory iterations.

Kanban implementation has been relatively successful in the manufacturing industry yielding various advantages. Successful histories of the manufacturing industry have convinced software engineers to adopt this approach; thus,

 ISBN: 978-1-61208-367-4

the adoption of Kanban in software development has gained strong practitioner-driven support.

The proliferation of the Kanban method in software engineering boomed after the first books were published [10][12]. The most popular of these books include David J. Anderson's "Kanban" [10], in which he introduced the concept of Kanban in systems and software development, and Corey Lada's "Scrumban" [12], in which he discusses the fusion of Scrum and Kanban.

A recent systematic literature review [6] on Kanban in software development shows certain benefits and challenges in the adoption of Kanban. The main benefits of using Kanban in software development are a better understanding of the whole process, improved quality of team communication and coordination with stakeholders, and better coverage of customer satisfaction [14][15][16]. According to [17], the use of Kanban in software development improved the lead-time to deliver software by 37%, the consistency of delivery rose by 47%, and defects reported by customers fell by 24% compared to the previously used Agile method. Because of the WIP limit, highest priority items are pulled to optimise value, resulting in improved customer satisfaction [14][15][17]. As Kanban does not involve any fixed plans, it helps to avoid requirement cramming.

According to [6], Kanban needs other supporting Agile practices to work effectively. However, this mixed approach has been problematic for many teams [18]. Motivating staff members to use Kanban has been challenging because of organisational culture and the stickiness of people to other familiar software development methodologies [14][16][18]. Additionally, a lack of specialised skills and training, and the misunderstanding of core principles have been reasons for failing to adopt Kanban [14][16][18].

III. RESEARCH SETTING

Empirical data was collected from software companies participating in a Cloud Software program. The empirical study was carried out in two stages. In the first stage, the Kanban usage survey was conducted among the participant companies' representatives. The survey was conducted in November 2013 using an Internet survey tool. The survey included questions on Kanban usage, motivation to use it, benefits achieved through its usage, and challenges faced in the Kanban adoption stage.

In the second stage, the survey results were complemented with semi-structured thematic interviews in December 2013, for which the survey respondents volunteered. The themes in the interview consisted of benefits and challenges of Kanban usage and methods to address and tackle the faced challenges.

In the first stage, an Internet survey request was emailed to the company representatives to identify and rate the importance of their usage motivations, the benefits achieved and challenges faced, while adopting Kanban in their unit. A five-point scale was used to assess the rate. The survey was open for two weeks. During this time, 21 persons, representing 10 different large software intensive companies, responded to the survey.

In the second stage, eight managerial level company representatives were interviewed. The purpose of the interviews was to complement the survey results by discovering the interviewees' views on Kanban usage in their teams or units. The duration of the interviews varied between 60 and 90 minutes (average 70 minutes). All interviews were recorded and transcribed by the interviewers, and the transcriptions were checked by the interviewees to ensure consistency.

IV. RESULTS

This section presents the results of the Kanban usage survey and the thematic interviews. The results of the survey are compared with results of earlier studies on Kanban.

A. Survey results

Table II and Table III present the positions of the respondents in their organisations and the respondents' experiences in software development.

TABLE II. RESPONDENTS BACKGROUND

Positions	**n**
President/CEO/CTO/Director	1
Program Manager/R&D Manager	3
Project Manager/Product owner/Agile coach	7
Analyst/Developer/Designer	8
Consultant/Trainer	2
Total	21

TABLE III. EXPERIENCE IN SOFTWARE DEVELOPMENT

Years of experience	**n**	**%**
1-5	4	19
6-10	1	5
More than 10	16	76
Total	21	100

The respondents were working in various positions in their organisations. The main organisational roles of the respondents were mid-level management (project managers, program managers, agile coaches, and analysts), and 76% had more than 10 years of software development experience. The typical sizes of working teams in the respondents' organisational units are presented in Table IV below.

TABLE IV. TEAM SIZE

Team size	**n**	**%**
1-3	1	5
4-6	5	24
7-9	11	52
10-12	3	14
13 and above	1	5
Total	21	100

The teams mostly comprise less than 10 persons. Only one company has teams with more than 13 persons.

Of the respondents, 57% reported that Kanban was used in their organisation. Most of the organisations using Kanban

 ISBN: 978-1-61208-367-4

had been using it for more than one year. The respondents of these organisations considered themselves competent in their knowledge of Kanban.

The Kanban users were requested to identify and rate the importance of motivation factors, achieved benefits and faced challenges while adopting Kanban in their unit. The results are presented in Tables V–VII.

TABLE V. KANBAN USAGE MOTIVATION

Motivation for choosing Kanban	Mean	Median
To improve team communication	4.4	5
To reduce development cycle times & time-to-market	4.2	5
To improve development flow	4.1	4
To increase productivity	4.1	4
To create transparency within the organisation	3.8	4
To improve product and service quality	3.8	4
To improve understanding of the whole value stream	3.5	4
To improve process quality	3.5	4
To improve stakeholders' satisfaction	3.3	4
To remove excess activities	3.3	3
To increase the ability to adapt to changes in the business environment	3.8	3
To decrease development costs	2.9	3
To improve organisational learning	2.6	3
To improve the management of business/ product value	2.5	3
To improve customer understanding	2.6	2

The results in Table V show the highest motivation factors in Kanban adoption are to improve team communication, reduce development cycle times and time-to-market, improve development flow, increase productivity, and create transparency within the organisation (with means of 4 or higher). According to [11], Kanban usage is motivated by its adaptability, ease of management by visualising the progress, and driving team members to cooperate and communicate. The motivation factors for using Kanban in software processes are aligned with what Anderson explained [6][10].

The respondents were also requested to indicate the achieved benefits of using Kanban. Those benefits are presented in Table VI.

TABLE VI. ACHIEVED BENEFITS OF USING KANBAN

Achieved benefits	Mean	Median
Better visibility of work	4.9	5
Improved transparency of work	4.5	5
Improved communication	4.3	4
Better flow controls	4.2	4
Better focus on your work	4.1	4
Better control on WIP	4.0	4
Enhanced efficiency	3.9	4
Better understanding of the whole process	3.7	4
Help in building trust	3.6	4
Help in predictability	3.6	4
Decrease in context switching	3.5	3
Enhanced quality	3.1	3
Assist leadership in strategic decision making	2.7	3

Table VI shows that the achieved benefits of using Kanban are similar to those in the literature. Better visibility, improved transparency of work and communication, better control of flow and WIP were the most common benefits experienced by the respondents of the survey. Such positive results correspond to the literature [4][6][7][10][12][19].

The results in Table VII show the order of significance of various challenges in Kanban adoption.

TABLE VII. CHALLENGES IN KANBAN ADOPTION

Challenges	Mean	Median
Lack of experience with the Kanban method	4.0	4
Hard to manage WIP limit	3.9	4
Hard to select tasks according to priority	3.3	3
Organisational culture was too traditional	3.2	3
Lack of knowledge and specialised training	3.1	3
No clear vision/roadmap for product(s)	3.0	3
Team members tend to fall back on using old methods	2.9	3
Teams were lacking decision-making ability & authority	2.6	3
Lack of customer/supplier collaboration	2.5	3
Unwillingness of team to follow Kanban	2.5	2
Incomplete planning for Kanban method adoption	2.4	2
Lack of management sponsorship	2.4	3
Decreased predictability	2.4	3
Burden of communication between teams or with in team	2.3	3
Inappropriateness of existing Kanban technologies/tools	2.3	3
Lack of customer presence	2.3	3
Lack of support from the management	2.2	2
Customer was not ready for increased communication	2.2	2
Incompatibility of business domain with Kanban method	1.9	2

 ISBN: 978-1-61208-367-4

Lack of experience with Kanban is the main challenge in adopting Kanban. Other major challenges are the difficulty managing the WIP limit and selecting tasks according to priority. Organisational culture, no clear vision or roadmap for product and team members tending to fall back on using old methods are other challenges that software companies face when adopting Kanban. All these challenges could be linked to one challenge: a lack of knowledge and specialised training.

B. Interview results

In this section, the results obtained from the interviews are described. The results of the thematic interview analysis can be classified into three main themes: benefits obtained, challenges faced, and solutions to the challenges in Kanban usage.

1) Benefits obtained

All of the interviewees reported that Kanban works well in their teams. Referring to the achieved benefits claimed in the survey, one interviewee mentioned, *"In our company, this is the default way of working in any team that has any support or operations type of work when the next two weeks just cannot be planned beforehand. I see the enhanced customer feedback loop as one of the main benefits".*

In line with the survey results regarding Kanban effects on their teamwork, an interviewee elaborated, "*Kanban works perfectly; the team can see the bigger picture of work and it brings realism to the work. Many tasks become clearer, especially high priority ones, and easier to justify, and we always find the balance between the demand and the capability. The team analysed their performance and velocity quite well, they understand the WIP limit and now they avoid taking too much work on*".

To explain the benefits of Kanban and the areas of successful implementation, one interviewee mentioned that in their "testing team", "*Kanban is beneficial because it puts limits on various things in each release and makes certain things visible which were not visible before Kanban adoption. Kanban makes work visible inside and outside the team and visualises customer needs. It helps in collecting new kinds of issues and a good example is the test area*".

All the interviewees reported that, with Kanban usage, the essence of collaborative work is visible and more and more work is collectively completed in the development team. Interviewees reported, "*Kanban helps to make bonds in teams and to start getting things done successfully. Kanban helps those team members who are in trouble to get the work done collaboratively*". Additionally, "*the team starts looking to where more and more things are actually waiting, and they start identifying root causes of why the flow is not working and make those blockers visible. To maintain the flow, the whole team starts solving them".* Furthermore, one interviewee mentioned, "*Kanban helps with flexibility when organising the work and efficiency, and so on*".

2) Challenges in Kanban usage

One of the bigger challenges found in the survey was the lack of experience using Kanban, which leads to some other challenges, e.g., difficulty managing the WIP limit and selecting tasks according to priority. An interviewee explained these challenges in this way: "I*n a company, not all business lines and top level management are familiar with Kanban. There are a few people who have knowledge but they also like to build their confidence to use the new Kanban approach. So, there is much resistance to change. The company lacks awareness about the existing mind-set issue. For example, for us, the release cycle is quite big; we are dealing with a huge requirement. In such a scenario, we need confidence that this can be done and delivered efficiently with this new Kanban thing. Nobody is willing to take the risk and start doing things the Kanban way. Everyone in the company knows that if we lose one release, we are out of the market, which means there's no company. The risk level is clear. It makes you feel safer to stay with what you already have.*"

Additionally, an interviewee agreed and reported "*The lack of training is a big challenge while using Kanban at both the portfolio and team level. The purpose and theory behind Kanban is misunderstood. The common question is raised, what is the problem we are trying to solve with Kanban?*" That is the reason why, when Kanban was implemented in the work without proper knowledge and training, an interviewee mentioned, "*Many times you don't actually necessarily look at the flow (of inflow and outflow of things) and the team don't actually use the WIP limits. When asking from where these WIP limits come, they guessed that the WIP limits were set without any planning.*"

Apart from knowledge and proper training, mind-set is equally important for the adoption of Kanban. With limited knowledge and experience, it is hard to motivate and change people's mind-sets to work using the Kanban method. This is also a reason for teams reverting to their previous way of working. Most of the companies interviewed agreed on this issue. One interviewee mentioned, "*People always want to stick with the way they work; it requires a lot of effort to change the mind-set*".

3) Solutions to challenges

The interviewees were asked what kind of solution they would propose or use to tackle the challenges they are facing in adopting Kanban. From the study, the following solutions were obtained to cope with the challenges:

- Provide proper training to the teams.
- Allow teams to experiment or pilot the method and get some experience using Kanban. Such piloting helps to learn the Kanban way of working by doing.
- Educating people help to change the mind-set so that the resistance to change will be easier to tackle. When people are educated and the expected benefits of Kanban are communicated, they will more likely be convinced to adopt it in their work.

 ISBN: 978-1-61208-367-4

- Commitment and awareness is required from the top-level management down through the company.

V. CONCLUSION

The goal of this study was to investigate the adoption and usage of Kanban in large software companies. The study was conducted in large multinational Finnish companies that have premises in several other countries. Using a survey and thematic interviews, the study aimed to analyse the motivations of Kanban usage, the obtained benefits, and the faced challenges when adopting Kanban. Additionally, solutions to the challenges were identified.

In this study, the identified main motivation factors for adopting Kanban were to improve team communication, and development flow, reduce time to reach the market, increase productivity, and create transparency in organisation. Furthermore, the most common achieved benefits of using Kanban were better visibility of work, improved transparency of work and communication and better control of flow.

Regarding the challenges faced, the most common one is lack of experience with Kanban, which makes it hard for teams to manage WIP limits and select tasks according to priority. Furthermore the traditional culture of the organisation was reported a challenge in the adoption of Kanban. Because of these challenges, team members tend to fall back on using old methods or previous ways of working. The literature and the results indicated that with proper training, the challenges could be handled to some extent.

The study subjects were representatives of large Finnish software companies, which provide a good general view of Kanban usage in advanced software companies utilising Agile and Lean approaches. However, the subjects of this study represent only a limited view of the participant companies. Therefore, to study more comprehensively the usage of Kanban throughout organisations, more extensive research involving all teams using Kanban in the organisation should be conducted. Future investigation would also be needed to gain a better understanding of the application of Kanban at different levels in the organisations, e.g., at the portfolio and team levels.

Despite a limited view of the participant companies, this study provides valuable descriptive information about the contemporary state of Kanban usage in software companies. The contributions of this paper are 1) up-to-date results on the current state of Kanban usage, based on first-hand industry insight on why Kanban is being used in software development as well as its benefits and challenges, 2) the first explorative study analysing the Kanban usage in software development, and 3) grounding for future research based on the identified main benefits and challenges on using Kanban.

ACKNOWLEDGMENT

The authors would like to thank the companies and their employees for participating to this research. This research has been carried out in Digile Need for Speed program, and it has been partially funded by Tekes (the Finnish Funding Agency for Technology and Innovation).

REFERENCES

[1] M. Poppendieck and T. Poppendieck, Lean Software Development: An Agile Toolkit, Addison-Wesley Professional, 2003.

[2] J. Liker, The Toyota Way, USA: McGraw-Hill, 2004.

[3] K. Hiranabe, Kanban applied to software development: From agile to lean, InfoQ. [Online] Available from: http://www. infoq. com/articles/hiranabe-lean-agile-kanban 2014.05.04

[4] A. Shalloway, B. Guy, and R. James Trott, Lean-agile Software Development: Achieving Enterprise Agility, Pearson Education, 2009.

[5] Cloud Software Finland, General Brochure. [Online] Available from: http://www.n4s.fi/en/ , http://www.cloudsoftwareprogram.org/general-brochure 2014.05.13

[6] M. O. Ahmad, J. Markkula, and M. Oivo, "Kanban in Software Development: A Systematic Literature Review," 39th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2013, pp. 9-16.

[7] L. Chai, "E-based Inter-enterprise Supply Chain Kanban for Demand and Order Fulfilment Management," International Conference on Emerging Technologies and Factory Automation, 2008, pp. 33-35.

[8] M. Becker and H. Szczerbicka, "Modeling and Optimization of Kanban Controlled Manufacturing Systems with GSPN including QN," International Conference on Systems, Man, and Cybernetics, 1998, vol. 1, pp. 570-575.

[9] M. Ikonen and P. Abrahamsson, "Anticipating Success of a Business-Critical Software Project: A Comparative Case Study of Waterfall and Agile Approaches", In Proc. ICSOB, 2010, pp. 187-192.

[10] D. Anderson, Kanban – Successful Evolutionary Change for Your Technology Business. Blue Hole Press, 2010.

[11] H. Kniberg and M. Skarin, "Kanban and scrum – Making the most of both," InfoQ, 2010.

[12] C. Ladas, Scrumban – Essays on Kanban Systems for Lean Software Development, Modus Cooperandi Press, 2009.

[13] J. Boeg, Priming Kanban, A 10 Step Guide to Optimizing Flow in Your Software Delivery System, Trifork Agile Excellence Mini Book Series, 2nd ed., Denmark at Chronografisk A/S, 2012.

[14] M. Senapathi, P. Middleton, and G. Evans, "Factors Affecting Effectiveness of Agile Usage – Insights from the BBC Worldwide Case Study," In Proc. XP, 2011, pp. 132-145.

[15] M. Ikonen, E. Pirinen, F. Fagerholm, P. Kettunen, and P. Abrahamsson, "On the Impact of Kanban on Software Project Work: An Empirical Case Study Investigation," In Proceedings of ICECCS, 2011, pp. 305-314.

[16] N. Nikitina and M. Kajko-Mattsson, "Developer-driven big-bang process transition from Scrum to Kanban," In Proceedings of ICSSP, 2011, pp. 159-168.

[17] P. Middleton, and D. Joyce, "Lean software management: BBC Worldwide case study," IEEE Transactions on Engineering Management, 2012, vol. 59(1), pp. 20-32.

[18] V. G. Stray, N. B. Moe, and T. Dingsøyr, "Challenges to Teamwork: A Multiple Case Study of Two Agile Teams," In Proceedings of XP, 2011, pp. 146-161.

[19] M. O. Ahmad, K. Liukkunen, J. Markkula, "Student perceptions and attitudes towards the software factory as a learning environment," In Proceedings of IEEE EDUCON, 2014, pp. 422,428.

 ISBN: 978-1-61208-367-4

Scaling Agile Estimation Methods with a Parametric Cost Model

Carl Friedrich Kreß
Scientific Services
Cost Xpert AG
Augsburg, Germany
carl.friedrich.kress@costxpert.de

Oliver Hummel
Software Design and Quality
KIT
Karlsruhe, Germany
hummel@kit.edu

Mahmudul Huq
Scientific Services
Cost Xpert AG
Augsburg, Germany
mahmudul.huq@costxpert.de

***Abstract*— Estimating the likely cost of a software development project is important with every process model. In agile settings, story points have proven being a useful tool to predict effort for small and medium sized projects or a few iterations. However when projects grow larger, their effort usually grows faster than a linear projection with story points would suggest. This can be attributed to so-called diseconomies of scale, e.g., caused by the growing communication overhead and need for refactoring in large projects. Although these effects are supported by all long-established parametric cost models, such as COCOMO, they are not yet taken into account with agile story point estimation. In this paper, we show how to calculate the magnitude of these non-linear effects to create awareness for this problem in the agile community. As a remedy, we propose three solutions to combine story points with COCOMO II in order to create advanced estimation methods that can be applied to large agile projects.**

Keywords-software cost estimation; COCOMO; agile; Scrum.

I. INTRODUCTION

Agile development approaches were initially aiming on software projects with manageable size and complexity. Due to their iterative and priority-driven implementation approach, they have become popular in many organizations. Even though – to our knowledge – there has been no scientific evidence that agile development projects are more successful than traditional approaches [1] so far, agile methods – like every other method or technique in software engineering – only seem to be helpful when conditions are right [2]. Nevertheless, it is not yet clear whether their perceived success is caused by increased development efficiency or just by the ability to steadily deliver working increments of a system under development. Nevertheless, this perceived success after decades of failed waterfall projects has raised the demand for scaling agile development approaches for larger undertakings. The scaling of agile projects is usually done organically, i.e., in a stepwise manner by splitting one team after a sprint to form the nuclei for two new teams that can then be filled with new additional people. As often reported in literature (see, e.g., [3]), this approach seems to work reasonably well in practice.

At the time being, agile approaches seem quite successful when it comes to estimating and planning two or three sprints ahead by analyzing the remaining user stories with the next highest priorities. Estimating the effort for complete agile projects, however, is a non-trivial challenge for various reasons. Many agile practitioners hence argue that it does not make sense to estimate a moving target (i.e., steadily changing requirements), but advocate to utilize a best effort design to cost approach that delivers as much functionality as the budget allows, billed on a time and material basis. Clearly, however, this is not satisfying from a management and controlling point of view: thus, it has led to the recommendation to elicitate and analyze more requirements in early iterations than can be implemented in order to quickly gain a coarse overview after a project has started [4]. Assuming that the size of each user story has finally been estimated in so-called story points [14], this would allow the prediction of a project's overall effort with the help of a burndown chart as soon as an initial velocity of the development team has been established after some initial sprints. The underlying estimation approach is similar to so-called expert judgments [9] that are a popular estimation method in non-agile environments.

From the perspective of large projects, however, both approaches suffers from a severe limitation that has only rarely been considered so far, especially in agile contexts: story points (as well as expert judgments) are a form of bottom-up estimation that predicts the overall project effort based on a linear projection ignoring so-called *diseconomies of scale*. The latter term describes the fact that larger development projects usually require disproportionally more effort than smaller projects, which can mainly be attributed to the growing communication and coordination overhead in larger undertakings [5]. Thus, although story-point-based estimation has proven to work reasonably well for projects of manageable size, it comes with significant drawbacks when it should be put to use in larger development efforts. Another issue that has recently been reported by practitioners is the increasing amount of refactoring required in growing agile projects. Although common sense clearly suggests that an incrementally extended system will require regular refactorings in order to remain maintainable and extensible, this continuously increasing technical debt [6] is ignored by the current, linear effort prediction via story points.

In order to highlight and overcome these limitations, the remainder of this paper is organized as follows: after going into more detail on the problem of diseconomies of scale and technical debt in Section II, we propose a set of three enhancements for agile estimation in Section III that will support agile developers in overcoming this challenge. The basic idea is to use some mathematics of the parametric estimation method COCOMO II in combination with the story point method to achieve more reliable effort estimates. Since our proposals can be used in different project contexts,

 ISBN: 978-1-61208-367-4

we briefly highlight their intended area of application in Section IV before we conclude our paper in Section V.

II. BACKGROUND

Various surveys have shown that even companies that adopted agile development methods still rely on traditional upfront project estimation and planning in many cases for business reasons. This is mainly because of the need to provide a project budget and status reports to customers or middle and top management, as, e.g. discussed by Sillitti and Succi [7]. We suppose that this necessity is not going to change anytime soon since project management standards like, PRINCE2 [17], that demand for a business justification, i.e., a cost-benefit analysis become more and more mandatory. Hence, even if the "domestic policy" of a development team is a settled agile methodology, in large enterprises and most customer relationships there will always be the need for a plan-based "foreign policy" justifying the expected effort to stakeholders outside the development team. The same need for planning in advance holds true from a strategic point of view. Assume, for example, that a company wants to evaluate a time-to-market strategy for a certain product. This again underlines the necessity for an estimation that quickly enables strategic planning before or at least soon after project start.

The practical need for dependable estimation in large agile projects is also enforced by emerging agile development models like the so-called *agile fixed price.* The clue is already in the name: in this model, a fixed price is agreed upon by suppliers and customers before or soon after the project is started [8]. Obviously, in order to be able to fix a price, the entire project scope must be determined in advance. Within the fixed price project the customer can then still decide what parts of the whole Information Technology (IT) product are to be developed with higher priority in an agile manner. This combination allows minimizing risks by setting a clear scope while at the same time providing a flexible –that is agile– project environment.

A. Diseconomies of Scale

As mentioned previously, agile estimates for the whole product backlog are relying on a linear effort projection: Agile teams measure how many story points they can deliver within a sprint and how much effort is required to do so. If, for example, a team can deliver 50 story points with 10 Person Month (PM) of effort, it can be concluded that, e.g., 300 remaining story points will roughly require 60 PM. Of course, one needs to steadily live with the risk that changing or misunderstood requirements will permanently disrupt this prediction and hence, most agile practitioners limit their estimations on the next two or three sprints. However, especially the management in larger organizations, usually, requires an upfront or early estimation of the whole project effort. The important aspect from an estimation point of view is that the pragmatic approach described above fully ignores the non-linearity of the size-effort relation in large software development projects, as already pointed out by Brooks [5] and Boehm [13]. This so-called diseconomy of scale has been confirmed subsequently by the regression analyses of every major parametric cost estimation model in use today, such as COCOMO II, REVIC, PRICE, and SEER [10]. However, it has to be acknowledged that there has been some controversy around this issue (see, e.g., [11]). Results that indicate slight economies of scale in smaller projects [12] are reflected in the COCOMO II model, which allows exponents smaller than 1 (see next section). But, even if such economies of scale can be reached in smaller projects, this amplifies schedule risks when projects need to scale as it could mean switching from economies of scale to diseconomies of scale.

The following Figure 1 demonstrates how an "over-linear" effort increase in large projects can indeed become a severe risk for the accuracy of agile effort estimations. It illustrates this graphically by contrasting a linearly growing effort curve (red lines), where effort is growing proportional to the expected size, with a nominal effort curve (shorter purple lines) calculated with Boehm's COCOMO II model [9]. Moreover, we have added two curves showing COCOMO II estimates under more and less complex project settings.

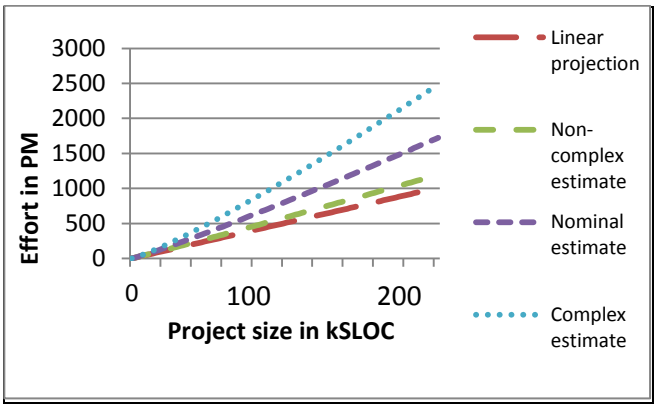


Figure 1. Linearly projected effort estimate against non-linear estimates using COCOMO II.

Clearly, in large projects the inherent empirical process control of agile methods would detect a decrease of development velocity over the course of the project (or at least the increasing refactoring efforts that have been reported by many practitioners once the code base has reached a significant size) and hence will better approximate the real effort over time. However, as described above the driver for estimation is the need to look ahead into the future for a significant amount of time and to present a realistic estimate for the overall effort expected for a project. This is where agile estimation as just presented has its weaknesses, especially when projects become larger. Ignoring this non-linear effort growth may lead to a dangerous underestimation of effort and in turn project duration that can endanger at least the business case of a project, if not the whole project itself.

 ISBN: 978-1-61208-367-4

B. Parametric Cost Models

Parametric cost models, like COCOMO II that we exemplarily present in this section, are based on a regression analysis of numerous projects. They usually require the estimated size of a project under consideration in kilo Source Lines of Code (kSLOC) and various complexity factors as input parameters. As soon as these are determined, the following formula (1) can be applied to calculate the expected development effort in person months:

$$\mathrm{PM} = \mathrm{A}*\mathrm{kSLOC}^{\mathrm{E}}*\prod_{\mathrm{i}=1}^{\mathrm{n}}\mathrm{EM}_{\mathrm{i}} \quad (1)$$

with *A* being a calibration constant describing the productivity, the expected size of a system is usually derived with the help of a functional size measure for the requirements, such as Function Points [9]. The other factors need to be determined by analysts from a project's context. Values and explanations for the Scale Factors (SF) required for calculating *E* and the Effort Multipliers (EM) can be looked up in the COCOMO II model definition [9].

However, function points, the conversion to Lines of Code, and the determining of the project parameters all bear an inherent inaccuracy so that estimation is also not a trivial task for traditional (i.e., non-agile) development approaches. As is visible in formula (1), COCOMO directly reflects non-linear growth through the exponent E (which is usually larger than one, but can also be slightly smaller, cf. Banker et al. [12]). Moreover, it also distinguishes between effort caused by the functional size (in kSLOC) with its exponent E on the one hand and the effort caused by the product of various so-called EMs, on the other hand. The effort multipliers represent the difficulty caused by non-functional requirements such as reusability needs or constraints in execution time as well as cost drivers such as overall product complexity or a desired internationalization. This distinction is nevertheless important since COCOMO II assumes that effort caused by the functional size grows in a non-linear fashion while the effort multipliers (although they themselves are discrete values) have a linear effect on the effort projection, as they just multiply the effort without an exponent.

As mentioned before, COCOMO II requires rating several scale factors that are deemed responsible fordiseconomies of scale. The following list gives a brief and simplified summary of these ratings:

- Precedentedness: rates if the product or project type is similar to previous ones.
- Development Flexibility: rates the software conformance to requirements and external interfaces.
- Team Cohesion: accounts for communication overhead because of difficulties in synchronizing stakeholders.
- Process Maturity: rates the maturity of the development process according to CMMI levels.
- Architecture / Risk Resolution: rates the maturity of the risk management concerning development risks as well as the percentage of development schedule devoted to establishing the software architecture.

Even though COCOMO II [9] was not developed with agile projects in mind, especially the last parameter reflects a circumstance that all software development projects do have in common, and that agile project are especially prone to: not putting (enough) upfront effort into the development of a decent software architecture can drastically increase the technical debt of a project and will increase refactoring overhead over the course of the project.

C. Error Calculation

The COCOMO II model can also be used to calculate the magnitude of error of a story point estimate like it was depicted in Figure 1 before. For that purpose, we assume that the functional size of the project simply increases by an arbitrary factor g, for the moment. Thus, the linear model used by agile teams would estimate the expected effort as:

$$\mathrm{g}*\text{Story Points}/\text{ Velocity} = \mathrm{PM}_{\mathrm{new}} \quad (2)$$

Here, velocity is given as story points per person month. There is no normalization factor that describes how to measure a single story point. This means, the number of story points can be of arbitrary size, depending on the habits of the agile team and it will only become meaningful when set in relation to person month needed per story point, thus describing the velocity of a specific team [15]. On the other hand, integrating the growth factor *g* into the COCOMO II formula causes a non-linear effort increase. This is shown in the following formula: since *g* is multiplied to the size it is under the influence of the exponent *E*:

$$\mathrm{A}*(\mathrm{g}*\mathrm{kSLOC}_{\mathrm{ref}})^{\mathrm{E}}*\mathrm{EM} = \mathrm{PM}_{\mathrm{new}} \quad (3)$$

To better illustrate the difference, we calculated the error as the fraction between both calculations. A 10 PM reference project would match a functional size of about 3.25 $\mathrm{kSLOC}_{\mathrm{ref}}$. This value is determined by "backward calculation" of COCOMO II for an effort of 10 PM, with scale factors set to high and all effort multipliers set to nominal, see (5). If, for example, the functional size growth factor between the reference project and another one is $g = 10$ the difference between the linearly interpolated estimate of 100 PM and the non-linear nominal scaling estimate equates to:

$$\frac{2.94*\left(\frac{100}{10}*3.25785\right)^{0.91+0.01*18.97}}{100} \approx 36\%. \quad (4)$$

In other words, the calculation in (4) demonstrated that even for a relatively small project a story point based effort prediction is prone to underestimate effort about one third.

III. Scaling Agile Estimation Methods

In this section, we propose three solutions with increasing accuracy to better represent the growing

communication overhead in large agile projects. They all work by combining the parametric cost model COCOMO II with common agile estimation practices. As such they are simply based on the common agile artefacts like user stories and story points. However, since there is no absolute size reference for one story point, it is not possible to simply use absolute story point counts in the formulas that we are suggesting in the following three solutions. In order to circumvent this problem, we have to work with the *relation* between story points and not the story points themselves. The same issue and solution must be considered for even simpler user story based estimates.

All presented solutions make use of the following data points. This reference data can be gathered during regularly sized projects (or initial sprints *before* scaling up the team) and allows determining the regular productivity of the team. In the following Sub-sections III.A, III.B and III.C, we will show three ways how to use this information to calculate the productivity of the upcoming larger project, that is, when the overall team size is scaled up. The following reference data is needed to determine the initial productivity:

- Story points delivered,
- Number of user stories and
- Effort in person month needed or kSLOC written.

A. Analogy-based Estimation using the Number of User Stories

If only a ballpark figure is needed (e.g., early in a project), we suggest the following simple approach to derive a coarse estimate that is merely based on the number of user stories and an analogy to a previous project or an initial increment. Using the COCOMO II formula presented above and the effort actuals of the previous project/increment, a functional size analogue for the functionality that the team delivered before can be derived by rearranging the COCOMO II formula using kSLOC:

$$kSLOC_{ref} = \left(\frac{PM_{ref}}{A*EM_{ref}}\right)^{\frac{1}{E_{ref}}} \quad (5)$$

When we consider the separation between growth caused by functional size and effort multipliers as explained before, this approach can only be applied under the following circumstances: 1) The user stories of the reference and the current project are of comparable size, that is their size differences are small for both projects. 2) The stories for the upcoming project are written in the same manner as in the reference project, especially in terms of the average size of a user story. 3) The effort multipliers do not change between the reference project and the current project, which is implicitly given when the reference data is coming from the same project.

Usually, these conditions will hold true when the upcoming project refers to the same class of products as the last project, e.g., when building a company's standard product like an interactive web application merely for a different customer. In these cases it is usually not necessary to rate the effort multipliers again. Combined with the assumption that the user stories are similar in size, the ratio between the number of user stories of the reference project ($\#\mathrm{US}_{\mathrm{ref}}$) and the number of user stories in the upcoming project ($\#\mathrm{US}_{\mathrm{new}}$) would then represent the change in functional size:

$$\mathrm{kSLOC}_{\mathrm{new}} = \mathrm{kSLOC}_{\mathrm{ref}} * \frac{\#\mathrm{US}_{\mathrm{new}}}{\#\mathrm{US}_{\mathrm{ref}}} \quad (6)$$

This calculated value can then be used for effort estimation with the help of the COCOMO II equation:

$$\mathrm{PM}_{\mathrm{new}} = \mathrm{A}*\mathrm{kSLOC}_{\mathrm{new}}{}^{\mathrm{E}_{\mathrm{new}}} \quad (7)$$

Although we are omitting a potential change of effort multipliers between the reference project and the upcoming project for sake of a quick estimate (i.e., $EM_{\mathrm{new}} = EM_{\mathrm{ref}}$), we do take the non-linear scaling factors (SF_j included in E) into account that are responsible for non-linear effort growth.

B. Analogy-based Estimation using Story Points

In order to improve the accuracy of the previous approach, it should be obvious that user stories weighted with story points produce better results as they also take size and complexity of the requirements into account. Concerning the distinction between functional size and effort multipliers discussed before, we can assume that a story point estimate is actually an amalgam of the functional size of the IT product and effort multipliers corresponding to the IT product.

Since the agile team judges effort multipliers implicitly when assigning story points, COCOMO II's effort multiplier ratings can be used to make this explicit as explained in the following. First, in order to eliminate this effect from the story point estimate and to gain a value for the pure functional size of the product we need to determine the effort multipliers [9] and "remove" them from the story point value through the following division:

$$\text{Functional size} = \frac{\text{Story Points}}{EM} \quad (8)$$

Second, based on these considerations we suggest the following steps to combine the "purified" story point estimate with the COCOMO II model in order to gain a reliable estimate for larger projects that also incorporates non-linear scaling effects:

1. Determine the functional size a team is able to deliver using kSLOC by backward calculation of COCOMO II (5).
2. Again it is necessary to determine how the functional size of the upcoming project changes in relation to the reference project. Thus, in order to merely relate the functional size of both projects with each other, we need to eliminate the effort multipliers from both story point estimates. The new functional size can then be derived as:

 ISBN: 978-1-61208-367-4

$$\mathrm{kSLOC_{new}} = \left(\frac{\mathrm{SP_{new}}}{\mathrm{EM_{new}}} \,/\, \frac{\mathrm{SP_{ref}}}{\mathrm{EM_{ref}}}\right) * \mathrm{kSLOC_{ref}} \qquad (9)$$

3. Now, these values can be used to calculate the expected effort, this time including the effort multiplier rating for the new product EM_{new} as well as a new evaluation of the scale factors E_{new} depending on the new team constellation and product environment:

$$\mathrm{PM_{new}} = \mathrm{A} * \mathrm{kSLOC_{new}}^{\mathrm{E_{new}}} * \mathrm{EM_{new}} \qquad (10)$$

C. Parametric Estimation Measuring SLOC

The previous two approaches are simple in the regard that they only use parameters well known to every agile developer and some algebra. However, the "backward" calculation (5) of the functional size of the reference system may lead to an additional estimation uncertainty that can actually be avoided. When reference code (or an initial increment of a new project) is available, it becomes possible to directly measure the size of the existing code base with some metric tool, ideally the COCOMO II code counting tool of Boehm's group at the University of Southern California [16]. This will increase the accuracy of the estimates with concrete numbers.

Based on such a concrete SLOC measure, it becomes possible to project the expected size of the new project again using the rule of three and the ratio of story points and effort multipliers as before (9). As shown above, it is then merely required to determine the scale factors and effort multipliers for the reference project and the new project. The result can then be easily used to estimate the overall effort required for the project by using the COCOMO II formula already used in (10). In other words, this third approach predicts the non-linear effort portion to be expected in a large development project with the COCOMO II model, based on a typical agile size measurement with story points.

IV. DISCUSSION

As the number of user stories is usually available before a concrete story point estimate, the method from III.A can be used in quite early stages, perhaps even before a project is actually started. When presenting the solution above, we suggested that the effort multipliers would not change between the reference project and the upcoming project. This makes sense as it might be difficult to rate the effort multipliers at such an early point in time. However, if time, resources, and the necessary information are available, this solution can even be refined by rating the effort multipliers and integrating them into the estimate as in the story-point-based solution (analogue to (9)).

The latter was designed with the goal in mind to be easily applicable by any agile team while providing good estimation results. It can be used after all user stories have been written and assigned a story point value. This, obviously, requires analyzing all user stories close to the beginning of a project even when they merely have a low priority. While analyzing more requirements than can be implemented in order to gain an overview of a project quickly is sometimes recommended in literature [4], many agile practitioners merely look ahead for two or three sprints and leave further stories untouched until they become relevant for short term planning. This approach obviously clashes with the business need of effort prediction. We do not see a simple solution for this dilemma, but regard the upfront analysis of user stories as a viable compromise that allows effort predictions without generating too much overhead.

Besides the value that the estimate provides from a business point of view for reliable product planning, we see an even higher value for agile teams: When asked to come up with an estimate, traditional agile estimation methods do not provide the means for anything else than a linear scaling. Thus, early in a project when the team size is still small, agile teams may be trapped by the self-created benchmark without a chance to predict reduced productivity when the project is scaled up later. Using our solution they can make the diseconomies of scale transparent and understandable to management by rating the COCOMO II scale factors and using the non-linearity of this model.

As mentioned above, we currently assume that a direct SLOC measurement would yield the most promising estimation results (although this measure is admittedly not undisputed itself it is probably the most accurate approach that is available today). This is because the SLOC that have been delivered during a reference project or a number of reference sprints best describe the actual productivity of a team. In addition the SLOC value could for example allow gathering historical data to determine a mean productivity factor. It could thus also be used to do a full COCOMO II calibration as described by Boehm et al. [9] in order to match a team's productivity even better. Thus, the SLOC-based solution is probably best suited for advanced agile teams that want to further improve their estimation accuracy.

Moreover, since COCOMO II defines SLOC very carefully, it should be made sure that the tool used to measure the reference SLOC complies with this definition in order to reduce sources of potential deviations. Whether the measurement of SLOC conducted with an organization's code metrics tool largely differs from the original COCOMO II SLOC counting definition can easily be verified by calibrating it with values delivered by the COCOMO II counting tool mentioned in Section III.C at least once or by directly using the latter to measure the SLOC.

Another interesting question that should certainly become subject of future research is the question how "pure" story point estimates reflect the functional size of a user story or how far they are "polluted" with the extra functional effort multipliers identified in COCOMO II. In Section III.B we have made the latter assumption, however, a closer look in the COCOMO II manual [9] suggest that some may be implicitly considered during story point estimation and others may be ignored. Hence, we feel that even common story point estimation could benefit from explicit consideration or exclusion of these factors. As our mathematical solution only evaluates the change between the effort multipliers of the reference project and the upcoming project, our

 ISBN: 978-1-61208-367-4

model should fortunately not be directly affected by the outcome of this future work.

V. CONCLUSION

The Agile Manifesto's intention [18] was not to create a reliable estimation method. It was about values and work culture, thus hit the nerve of the time and has inspired several successful agile development approaches. However, basically all agile methodologies were initially aiming on smaller projects with small teams and only recently ideas for scaling them in a stepwise manner have been added. As we have described in this paper, even agile projects are often under a significant outside pressure to deliver reliable effort estimates. The larger projects, the larger this pressure will usually become. Exactly such larger software development projects have to deal with so-called diseconomies of scale caused by the growing need for communication and coordination amongst their personnel due to the growing size and complexity of the software system. This non-linear increase of development effort with project size, is not reflected in current agile estimation techniques based on story points and hence poses a serious risk of under-estimation for larger projects.

In this paper, we described three advanced ideas to better deal with this challenge by combining agile estimation techniques with elements from the proven parametric cost model COCOMO II, as initially developed by Barry Boehm. Although in this early stage, the ideas look promising; it is obvious that the next step must be an investigation of their practical relevance. To our knowledge, there is no study that would have looked into the non-linear effort increase in large agile projects and hence no empirical data is readily available that could be used to validate our model. However, well managed agile projects that have tracked their development efforts should allow applying all three proposed approaches in retrospect so that predicting their overall effort based on the velocity of, e.g., the first three sprints and COCOMO II should become possible.

Even though a lot of work still needs to be done, we conclude that the combination of agile estimation methods and parametric cost models can be seen as a promising way for agile estimation in the 21st century software engineering that might help better predicting the growing communication and refactoring overhead in large agile projects.

VI. REFERENCES

[1] T. Dyba and T. Dingsøyr, "Empirical studies of agile software development: A systematic review.", Information and Software Technology, vol. 50, iss. 9–10, August 2008, pp. 833-859.

[2] T. Chow and D.B. Cao, "A survey study of critical success factors in agile software projects", The Journal of Systems and Software, vol. 81, iss. 6, June 2008, pp. 961-971.

[3] Paasivaara, S. Durasiewicz, and C. Lassenius, "Distributed agile development: Using Scrum in a large project", Proc. IEEE International Conference Global Software Engineering, 2008, pp. 87-95.

[4] C. Larman, Applying UML and Patterns, Prentice Hall, Upper Saddle River, 2005.

[5] F. Brooks, The Mythical Man Month, Essays on Software Engineering, Addison-Wesley, 1975.

[6] P. Kruchten, R.L. Nord, I. Ozkaya, "Technical Debt – From Metaphor to Practice", IEEE Software, vol. 29, iss. 6, November/December 2012, pp. 18-21.

[7] A. Sillitti and G. Succi, "The Role of Plan-Based Approaches in Organizing Agile Companies", Cutter IT journal, vol. 9, iss. 2, 2006, pp. 14-19, URL: http://goo.gl/CYRvJK (retrieved: January 2014).

[8] A. Opelt, B. Gloger, W. Pfarl, and R. Mittermayr., Agile Contracts: Creating and Managing Successful Projects with Scrum, Hoboken, 2013.

[9] B.W. Boehm et al., Software Cost Estimation with COCOMO II, Prentice Hall, Upper Saddle River, 2000.

[10] R. Jensen, A.W. Armentrout, and R.M. Trujillo, "Software Estimating Models: Three Viewpoints.", CrossTalk, February 2006, pp. 23-29, URL: http://goo.gl/ACfzDk (retrieved: January 2014).

[11] B.A. Kitchenham, "The Question of Scale in Software – why cannot researchers agree?", Information and Software Technology, vol. 44, iss. 1, January 2002, pp. 13-24.

[12] R.D. Banker, H. Chang, and C.F. Kemerer, "Evidence on economies of scale in software development", Information and Software Technology, vol. 36, iss. 5, May 1994, pp. 275-282.

[13] B.W. Boehm, Software Engineering Economics, Prentice Hall, 1981.

[14] R.L. Nord and J.E. Tomayko, "Software Architecture-Centric Methods and Agile Development", IEEE Software, vol. 23, iss. 02, March-April 2006, pp. 47-53.

[15] M. Cohn, Agile Estimating and Planning, Prentice Hall, Upper Saddle River, 2005.

[16] URL: http://sunset.usc.edu/research/CODECOUNT (retrieved: January 2014).

[17] Great Britain. Office of Government Commerce: "Managing successful projects with PRINCE2", TSO 2009

[18] URL: http://agilemanifesto.org/ (retrieved: January 2014).

 ISBN: 978-1-61208-367-4

AP3M-SW – An Agile Project Management Maturity Model for Software Organizations

Felipe Santana Furtado Soares
UFPE/CIn – Informatics Center – Federal University of PE
C.E.S.A.R - Recife Center of Advanced Studies and Systems
Recife, Brazil
furtado.fs@gmail.com

Silvio Romero de Lemos Meira
UFPE/CIn – Informatics Center – Federal University of PE
Recife, Brazil
srlm@cin.ufpe.br

***Abstract*—Applying agile methodologies in organizations whose processes are based on maturity models, such as Capability Maturity Model Integration (CMMI) or Organizational Project Management Maturity Model (OPM3) has been the focus of much controversy in the academic and in the software industry environment. The two approaches apparently have some fundamental principles and different bases, but on the other hand, adopting them jointly has increasingly become a reality for software organizations. However, the rush to reach maturity levels within deadlines that are shorter and shorter and the definition of heavy and inflexible processes, result in improvement projects with unique objectives of adherence to such models, often reflected in carrying out unnecessary activities and generating excessive documentation. In this context, agile methodologies are more appealing as they are lighter and this is inevitably related to their apparently offering a faster development at a lower cost of human effort. In this scenario, this paper puts forward a definition of an agile project management maturity model for software development organizations.**

Keywords—Project Management; Agile Methodologies; Maturity Model; APM; OPM3; CMMI.

I. INTRODUCTION

Currently, one of the challenges of software organizations is to acquire maturity in their development processes by means of implementing improvement projects based on recommendations of quality models recognized worldwide, such as Capability Maturity Model Integration (CMMI) [1].

At the same time, applying agile methodologies in organizations whose processes are based on maturity models, such as CMMI or the Organizational Project Management Maturity Model (OPM3) [2] has been the focus of polemical debate both in the academic world and in the environment of the software industry. The two approaches seem to introduce some fundamental principles and bases that diverge from each other but, on the other hand, adopting them jointly has become a reality for software organizations [3].

According to the *The Chaos Report* [4] between 2008 and 2010, the rate of projects categorized as 'Success' increased from 32% to 37%, while the rate of projects categorized as 'Cancelled' decreased from 24% to 21%. The rate of 'Challenged ' projects decreased from 44% to 42%. Among the reasons that the Report gives for this significant improvement found in 2010, in relation to 2008, the following can be highlighted:

- The use of agile processes has been growing. Currently, they represent 9% of all Information Technology projects and have been adopted in 29% of new applications under development. The Institute concludes that the growth in the rate of 'Success' is directly related to the increase in adopting agile methodologies;
- The reduction in the use of the processes that follow the 'Waterfall" lifecycle, known as traditional methods, has already accounted for nearly 50% of the number of new implementations. However, some companies are still having difficulties in implementing the methodologies, sometimes for lack of knowledge, sometimes because of the difficulty in adapting such methodologies to the context of their projects [6].

Nowadays, the competitive differential no longer lies in using such methodologies but rather in overcoming the challenges implementing them correctly and in the search for continuous improvement in software development processes [6]. Scrum, one of the methodologies that has gained most popularity, has been used in different ways, sometimes for lack of knowledge in its use, sometimes because it does not completely fit into the needs of companies. Other approaches are available, for example, the use of Kanban in software maintenance projects, in which features such as fixed iterations may not make sense for all projects [14].

Several studies report the adoption and growth of the use of agile methodologies in recent years. What can be perceived already is that organizations have consolidated their interest in them, the growth of the agile community, the high level of discussions, events, etc. [6].

Mike Cohn [6] states that seeking knowledge of agile methods has grown and that this cannot be considered a simple fad. However, what is observed is the difficulty that organizations have in implementing them, sometimes due to badly conducted adaptations, which strike at agile values and principles, sometimes due to the excessive restrictions that a methodology has and which cannot be fitted into the needs of certain projects.

According to Sidky *et al.* [15], it is observed that even with the growing number of companies that are seeking to adopt agile processes, there are still few studies that guide companies in this adoption. When organizations attempt to implement agile methodologies in a non-systematic way, projects end up having the same problems previously found in traditional methodologies.

 ISBN: 978-1-61208-367-4

At the same time, the adoption of maturity models in project management has been growing in the world [2]. However, none of them is exhaustively focused on implementing an agile project management in software development organizations, even the CMMI is sufficient with all processes well defined because it doesn't address agile methods directly. Some of the most widely used models for example are: OPM3 (Organizational Project Management Maturity Model) [2], KPMMM (Project Management Maturity Model) [22], CMMI (Capability Maturity Model Integration) [1], PMMM Project Management Maturity Model) [24], MMGP (Maturity Model for Project Management) [23] and P2MM (PRINCE2 Maturity Model) [24].

However, if there is a clear motivation for using methodologies that promote agility in development, the search for certifications and adherence to maturity levels continues. Accordingly, strategies that result in maturity of processes based on agile principles have been a common target among software companies.

In this context, this paper aims to answer the following *question-problem*: with a view to increasing the success rate of software development projects, is a maturity model effective as part of the organizational strategy of implementing agile project management gradually and in a disciplined way?

To answer this question, this paper presents a maturity model that can guide software development organizations in implementing agile project management projects based on the existing main maturity models (CMMI and OPM3), while making use of the best practices of APM (Agile Project Management) [17] and Agile Methods (Scrum [7], FDD – Feature Driven Development [12], Lean [16] Kanban [14], Crystal [8] XP – Extreme Programming [11]), in a disciplined and gradual manner.

The paper is divided as follows: Section 2 presents the background overview of project management maturity model, agile methodologies and agile project management; Section 3 presents an initial discussion about an agile project management maturity model, showing the benefits of agile methodologies and the model components; The last section concludes this work in progress and presents the future studies, including the model validation research methodology.

II. BACKGROUND OVERVIEW

A. Project Management Maturity Model

Over the years, organizations have been increasingly motivated to adopt quality models focused on the maturity of the software process. One of the reasons for this is associated with the fact that the improvement in the quality of software is widely associated with the adequacy and adherence of their processes to the high levels of this model [19].

Maturity may be defined as "*a form of measuring the stage of an organization's ability to manage its projects*" [23]. A maturity model, in accordance with OPM3 (2003) [2] is a conceptual framework, with consistent parts, which defines the maturity of an area of interest, for example, the organizational management of projects.

Figure 1 shows the timeline with reference to the main maturity models.

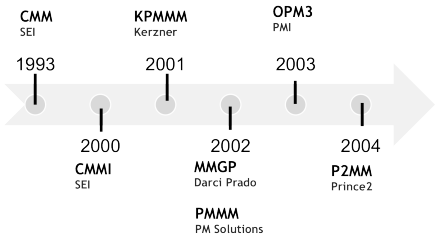


Fig. 1. Timeline with reference to the main maturity models.

The Capability Maturity Model (CMM) was originally developed by Watts Humphrey [20] and first appeared in his book 'Managing the Software Process'. He was inspired by the 20th century movement of manufacturing and quality assurance of the work of Juran, Deming and Crosby. The term "maturity model" and the five levels were inspired by Crosby's manufacturing model [28].

The CMMI is a process improvement approach that provides elements that are essential to an effective process. It brings together best practices that address development and maintenance activities, and covers the entire lifecycle of products from their conception to delivery and maintenance [1].

The KPMMM was created by Harold Kerzner and is set at five levels (Common Language, Common Processes, Unique Methodology, Benchmarking, and Continuous Improvement). It defines the current stage, planning and actions for implementing and gradually developing the management of projects [22].

The MMGP was created by Darci Prado and uses the same levels as the CMM Model. It possesses simplicity and universality (it is applicable to all types of organizations and to all categories of project) [23].

The PMMM of PM Solutions was created by Crawford also has five maturity levels and nine knowledge areas [24].

The OPM3 was established by the PMI (Project Management Institute). It is a model that aims to provide a path so that organizations understand their organizational management of projects and to measure the maturity based on a set of best practices in organizational project management. It describes a process in which the organization can develop or find a set of skills or good practices [2].

The P2MM was created by the Office of Government Commerce in 2006 and is based on the Project In Controlled Environments methodology [25].

B. Agile Methodologies

In the last decade, agile methodologies have been gaining space in the Information Technology and Communication market. Many studies show good results achieved by these companies, for example, research conducted by Scott Ambler reported a 55% success rate of projects, which used agile methodologies [26].

 ISBN: 978-1-61208-367-4

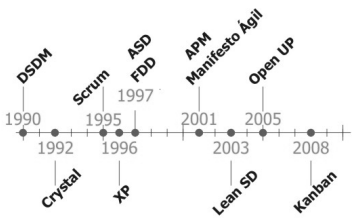


Fig. 2. Timeline referring to the main milestones which involve agile.

Figure 2 shows the timeline with reference to the milestones involving agile methods: Dynamic Systems Development Method (DSDM) [10], Crystal [8], Scrum [7], Extreme Programming (XP) [11], Adaptive Software Development (ASD) [9], Feature Driven Development (FDD) [12], Agile Project Management (APM) [17], the Agile Manifesto [27], Lean Software Development [16], Open Unified Process and Kanban [14].

Scrum is a framework established in 1996 by Schwaber and Sutherland and brings together monitoring and feedback activities, in general, in quick, daily meetings with the entire team, thus aiming to identify and correct any deficiencies and/or impediments to the development process [7]. Among the most used methodologies, Scrum appears as one of those that organizations most prefer (56%). In this same survey, the joint adoption with Kanban begins to be perceived [13].

XP was created by Kent Beck [11] in 1996 and seeks to enhance a software project using five essential values: communication, simplicity, feedback, respect and courage. Practices such as pair programming, rapid changes, constant feedback are core elements of the culture of this community.

FDD was created in 1997 in a large project in Java in Singapore. It arose from Coad's experience of object-oriented analysis and modeling, and Project management by De Luca. It is an agile methodology for managing and developing software, which combines the best practices of the agile management of projects with a complete approach to object-oriented Software Engineering [12].

Lean emerged in Toyota based on the idea that an increase in productivity is related to stopping doing anything that does not add value to the customer. Lean makes us think in a fast, uniform and quality flow without extra work that does not need to be done, without added defects [16]. Kanban brings the philosophy of Just in Time (JIT), which means producing only what is necessary, in the necessary time, in the necessary amount, and in the necessary location, and to do so with quality and the involvement of people, thus eliminating waste, and ensuring the continuous flow of production [14].

C. Agile Project Management (APM)

APM uses an empirical process model based on inspection and adaptation in order to promote exploration and an adaptive culture, to allow self-organization and self-discipline, to promote the reliability and consistency possible, given the degree of uncertainty and complexity inherent in the project, to be flexible and easily adaptable, allowing visibility throughout the process, to embed the learning, to encompass the specific practices of each stage and to provide points of verification [17].

According to Highsmith [17], *"[...] APM brings in itself a new focus on systems development, founded on agility, flexibility, communication skills and the ability to offer new products of value to the market, in short periods of time."*

The five stages of APM ***(Vision, Speculation, Adaptation, Exploration and Closure)*** were defined in order to promote the continuous delivery of value and to allow reflection that promotes learning. APM discards the anticipatory posture, based on prior planning actions and activities, characteristics of traditional project management, and seeks to develop a vision of the future and the ability to perform through situational exploration.

III. AP3M-SW – AN AGILE PROJECT MANAGEMENT MATURITY MODEL

According to Cohn [6], agile methodologies are generating significant gains in productivity with reductions in equivalent costs. This is due to several reasons: the adoption of these methods, which have mechanisms to release products on the market with much greater speed and to the satisfaction the client. In addition, they make it possible to visualize the development process better, which leads to greater predictability.

A. Benefits of Agile Software Development

Cohn [6] consolidated four surveys conducted in 2008 on the benefits of adopting the agile development of software related to the following aspects: productivity; time-to-market; and product quality: Mah [18] of QSMA, Rico [21], Version One [13] and Ambler [26].

Regarding the comparison on **productivity**, research by Mah [3] reports that agile projects are 16% more productive with a confidence level, which is statistically significant.

Regarding the **time-to-market**, agile teams tend to launch their products faster than traditional teams. VersionOne [13] reported that 64% of participants stated that the time-to-market improved (41%) or significantly improved (23%). Mah [18] compared 26 agile products to the QSMA database with 7,500 projects and showed that their time-to-market is 37% faster.

Regarding the **quality** of the product, Rico [21] states that agile teams develop higher quality products, based on 51 published studies on agile projects: a minimum improvement of 10% in quality and an average increase of 63%. According to Version One [13], 78% of participants responded that agile development improved (44%) or significantly improved (24%) software quality. In addition, 84% of the participants thought that the number of defects reduced by 10% or more.

However, according to Anderson [28], transition initiatives to agile methods may fail because prescriptive processes are powered by an organization to the delivery of the program as a part of the initiative and conducted by a process improvement group, an agile training group or a form of external consultancy. The workforce appears to tolerate the initiative, but actually passively resists this, because they believe that their unique situation does not fit into a standard process and the change is being forced, often without consultation or consensus.

In this scenario, the need therefore emerges to define a model that assists companies in implementing an agile

 ISBN: 978-1-61208-367-4

management of projects in a more structured and mature manner.

B. Model Components

The way to achieve greater agility with the maturity model is to realize that the practices are primarily advisory or indicative, and that to correspond to an evaluation, an organization must demonstrate that the goals of a process area are being reached through evidence practice [28].

In this context, and based on the main models, frameworks and methods listed below, this study defines a maturity model for implementing the agile management of projects, the AP3M-SW, with the following features:

- It is based on **CMMI** to define five project management process areas with their respective specific goals and practices of software development, namely: **Project Planning** (to establish estimates, develop a project plan and obtain commitment to the plan); **Project Monitoring and Control** (to monitor the project against the plan and manage corrective action to closure); **Requirement Management** (to manage requirements); **Risk Management** (to prepare for risk management, identify and analyze risks and mitigate risks); and **Integrated Project Management** (to use the project's defined process and coordinate and collaborate with relevant stakeholders);
- It is based on the **OPM3** to define the domains of organizational project management **(Project, Program and Portfolio)** and the stages at which the organization is to be found **(Standardized, Measured, Control and Continuously Improved)**;
- It is based on the **APM** phases so as to define the project management process groups: **Vision, Speculation, Adaptation, Exploration and Closure**;
- It is based on **Agile Methods** (Crystal, Scrum, FDD, XP, Lean, and Kanban) so as to define practices and work products of each of the process areas.

Figure 3 illustrates the main components of AP3M-SW.

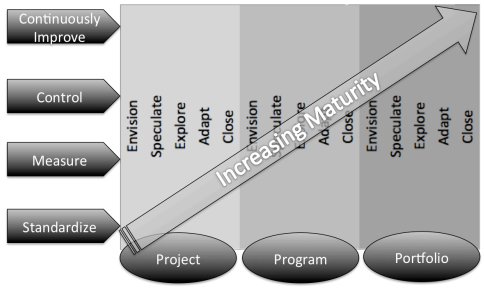


Fig. 3. Main components of the AP3M-SW Model (Adapted from [2]).

IV. CONCLUSION AND FUTURE STUDIES

The transition from traditional methods to agile methods and the changes necessary to obtain their real benefits are difficult to achieve. The change affects not only the software development team, but also various areas of the organization and, above all, this requires a cultural change.

Aligned to this context of the growth of the agile methods, the adoption of various maturity models in project management is growing worldwide. The challenge becomes how to combine these two approaches without losing their main features.

Various studies have already been conducted showing the possibility of getting on peacefully with agile and mature approaches [5]. If, on the one hand, it is possible to add practices of the maturity model not considered in agile methodologies, values and principles should not be compromised.

To guide companies who experience this scenario, this paper defined a maturity model that gives support in a disciplined and gradual manner when implementing agile project management based on relevant models, frameworks and already validated methods and on the community´s growing use of software development methods.

Future work is expected to detail all models' components and validate it. The main challenge of this validation is related to the possibility of applying the model in a software development company, through a case study methodology [29][30], and defining what metrics may be collected before and after adopting the model. Furthermore, isolating the variables before and after measurement to be able to assess if, indeed, the use of the model contributed to the success of implementing agile project management and, consequently, measuring the impact of this on the results of projects in relation to complying with the costs, time, scope, quality and satisfaction of the client and team. The time needed for implementation also presents a strong constraint, bearing in mind that maturity models need to be used gradually.

REFERENCES

[1] CMMI-DEV, CMMI for Development, V1.3 model, CMU/SEI-2010-TR-033, Software Engineering Institute, 2010.

[2] Organizational Project Management Maturity Model (OPM3®) Knowledge Foundation do Project Management Institute (PMI). 2003.

[3] F. S. F. Soares and S. R. L. Meira, "An Agile Maturity Model for Software Development Organizations", ICSEA, Venice, Italy, October, 2013, pp. 324-328.

[4] Standish Group. The Chaos Report. 2009, 2011. Available at: http://www.standishgroup.com. Last access: 08/2014.

[5] A. Marçal, B. Freitas, F. Furtado, T. Maciel, and A. Belchior, "Extending Scrum according to the CMMI Project Management Process Areas". CLEI, San José, 2007. Portuguese Version Only.

[6] M. Cohn, "Succeding with Agile: Software Development with Scrum", Bookman, 2011.

[7] K. Schwabber, "Agile project management with Scrum". EUA, Microsoft, 2004.

[8] A. Cockburn, "Crystal Clear: A Human-Powered Methodology for Small Teams", 1st edition, Addison-Wesley Professional, 2005.

[9] J. A. Highsmith, "Adaptive Software Development: A Collaborative Approach to Managing Complex Systems", New York, NY, Dorset House Publishing, 2000.

[10] J. Stapleton, "DSDM: Dynamic Systems Development Method: The Method in Practice", Addison-Wesley, Boston, MA, 1997.

[11] K. Beck, "Extreme Programming Explained: Embrace Change", 2nd Edition, Boston, MA, Addison-Wesley Professional, 2005.

[12] S. R. Palmer and J. M. Felsing, "A Practical Guide to Feature-Driven Development", NJ, Prentice Hall, 2002.

[13] VersionOne, "State of Agile Development Survey Results", Available at: http://www.versionone.com, 2008, 2013, Last access: 08/2014.

[14] D. J. Anderson, "Kanban – Succesful evolutionary change for your technology business". Blue Hole Press, Washington, 2010.

[15] A. Sidky, J, Arthur, and S, Bohner, "A disciplined approach to adoption agile practices: the agile adoption framework. Innovations In Systems

 ISBN: 978-1-61208-367-4

and Software Engineering, 2007, Volume 3, Number 3, pp. 203-216 (14).

[16] M. Poppendieck and T. Poppendieck, “Lean Software Development: An Agile Toolkit”, Addison-Wesley, Longman P. Co., Boston, MA, 2003.

[17] J. Highsmith, “Agile Project Management – Creating Innovative Products. EUA: AddisonWesley, 2004.

[18] M. Mah, “How agile projects measure up, and what this means to you”. Cutter Consortium Agile Product & PM Executive Report 9, 2008.

[19] D. Goldenson and D. Gibson, “Demonstrating the impact and benefits of CMMI: An update and preliminary results”, CMU/SEI-2003-SR-009. SEI. Pittsburgh, PA: SEI, 2003.

[20] W. S. Humphrey, “Managing the Software Process”. EUA: Addison-Wesley Professional, 1989.

[21] D. F. Rico, “What is the ROI of agile vs. traditional methods? An analysis of extreme programming, test-driven development, pair programming and Scrum”. TickIT International, 2008, 10(4), pp. 9-18.

[22] H. Kerzner, “Kerzner Project Management Maturity Model (KPMMM)”. Wiley, 2o edition. 2005.

[23] D. Prado, “Project Management Maturity Model (MMGP)”, Belo Horizonte: EDG, 2008. Portuguese Version Only.

[24] J. K. Crawford, “Project Management Maturity Model (PMMM)”, PM Solutions., 2o Edition, Center for Business Practices. 2007.

[25] PRINCE2 Maturity Model: Office of Government Commerce. 2006.

[26] S. Ambler, “Agile adoption rate survey”. Available at: http://www.ambysoft.com/surveys/, 2008, 2010. Last access: 08/2014.

[27] K. Beck et al. “Manifesto for agile software development”. Utah, 2001. Available at: http://agilemanifesto.org/. Last Access: 21/02/2012.

[28] D. J. Anderson, "Stretching Agile to fit CMMI Level 3", Agile 2005 Conference, Denver, July, 2005.

[29] B. A. Kitchenham and L. M. Pickard, “Evaluating software engineering methods and tools: part 9: quantitative case study methodology”. ACM SIGSOFT Software Engineering Notes, 1998, 23(1), pp. 24–26.

[30] R. K. Yin, “Case Study Research: Design and Methods”, EUA: Sage Publications, 2001, 200p.

 ISBN: 978-1-61208-367-4

Cybernetic Aspects in the Agile Process Model Scrum

Michael Bogner, Maria Hronek, Andreas Hofer, Franz Wiesinger

University of Applied Sciences Upper Austria
Department of Embedded Systems Engineering
Hagenberg, Austria
Email: [michael.bogner, maria.hronek, andreas.hofer, franz.wiesinger]@fh-hagenberg.at

***Abstract*—Agile process models provide guidelines for modern software development. As one of their main purposes is to complete projects under external influences as successfully as possible, the question arises as to how reliably and routinely given project goals can be achieved by means of such process models. This is all the more relevant as today, unfinished software projects frequently lack certain functionality, or missed project deadlines are still on the daily agenda in software development. Therefore, research has been done to identify the coherences between agile process models and cybernetics. Cybernetics is a natural science based on biocybernetics which forms the basis for well-functioning processes. It was analysed how it helps to cope with complexity, thus allowing for viable complex systems or processes. Cybernetics, as a science of functioning, is also relevant for agile process models. Once the basic cybernetic aspects are applied, processes are kept under control and organized in ways that ensure long-term viability. This paper reports the results of the selected agile representative Scrum. It shows that although some major cybernetic aspects like communication, feedback and circularity are covered, other basic cybernetic principles are missing in Scrum. Yet, these shortcomings can be compensated in order to get essential reliability, especially in critical situations.**

Keywords–agile software development; Scrum; cybernetics; biocybernetics.

I. INTRODUCTION

A major challenge of today's project management is the increasing complexity and the dynamics of changing conditions. Project requirements are getting more and more complex and therefore often cause serious problems for project managers. To meet these requirements in modern software development and to cope with complexity we use process models like the agile model Scrum [1]. It is easily understandable but not absolutely easy to use. In some cases, complex problems can result in loss of control [2]. To counteract this, we typically try to eliminate one problem after the other arising during the development cycle. But this leads to higher costs and missed project deadlines. As a result, the product quality is strongly affected and the project goals will be missed in some cases.

We often overlook that there is a science that can help to cope with complexity. It helps to lead a project in the right way from the beginning to the end. It shows how to survey the complexity and how to deal with it without fighting against it. This science is called cybernetics. It is a natural science which is the basis of many well-functioning processes. Processes and procedures can be kept under control with this science [3]. Cybernetics is an integrative multidisciplinary meta-science. It comprises various theories, primarily the theories of information and communication, and the theory of regulation and control. Without the laws of cybernetics, almost nothing would work - no aircraft, no computer, no large city and no organism [4]. As one of the most fundamental and powerful sciences, cybernetics incorporates the essential mechanisms in order to cope with complexity: self-control, self-regulation and self-organization. In our world of increasing complexity, cybernetics provides the invariant laws of functioning. This holds true for biological, technical, physical, social and economic systems, but most people are not aware of that [5] [6].

As cybernetics is the powerful meta-science which helps to accomplish complex processes successfully, the question arises, whether agile software process models fulfill the basic requirements of cybernetics to reliably guide the entire development cycle. This question addresses not just certain business cases, but is effectively a fundamental question. Cybernetics defines the basic laws, which have to be fulfilled otherwise the development process could get out of control [3]. We have started our research activities with the agile process model Scrum. Scrum is not only well-known, but also widely used. Therefore, we have selected it as our first research candidate. The cybernetic aspects in Scrum have been worked out and they will be described and discussed below to show if they keep the development process under control. The results are presented in this paper.

In addition, we have analysed the aspects of biocybernetics to see if the process model meets the requirements to be long-term viable. Whether certain processes or systems will be viable and capable of "surviving" permanently depends on how far they obey certain basic principles of biocybernetics [7]. These principles and the results of our research have been summarized and are also included in this paper.

This paper is divided as follows: Section II briefly presents the main aspects of Scrum being relevant for this paper. Section III introduces cybernetics in order to make a comparison afterwards in Section IV. Biocybernetics with its basic rules is introduced in Section V. Each rule is discussed with regard to Scrum. Finally, Section VI summarizes the relationship of Scrum and cybernetics and concludes the paper.

II. SCRUM

Scrum is a lightweight agile process model developed by Ken Schwaber and Jeff Sutherland [1]. It provides a framework to manage complex product development and it has clearly defined rules and regulations. The development

 ISBN: 978-1-61208-367-4

process remains flexible and transparent all the time. Scrum is based on iterations where each one usually lasts one to a maximum of four weeks. This iteration is called a Sprint. In the beginning, basic product requirements must be known and committed to the Product Backlog. The requirements are split into tasks and stored in the Sprint Backlog in order to start a Sprint. This process is shown in Figure 1. These tasks are the most important tasks which should be handled during the next iteration. The intention is to have a potentially executable product at the end of a Sprint, called the product increment. The functionality grows from Sprint to Sprint.

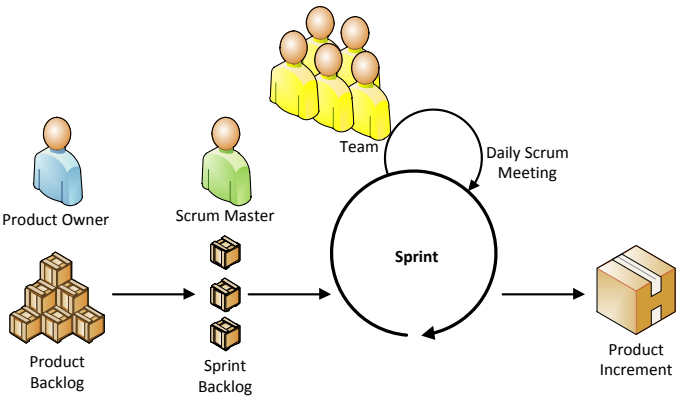


Figure 1. The Scrum Process

There are three essential roles that bring Scrum to life. The *Scrum Team* consists of a group of seven plus/minus two people. They execute the tasks from the Sprint Backlog. The team works in a self-organized fashion. Another important role is the *Scrum Master*. He or she has to see that impediments to the team's progress are removed. The Scrum Master also has to ensure that the process model is proceeded correctly. The *Product Owner* is the third defined role. He or she argues the project goal and defines and prioritizes the single work packages in order to maintain the Product Backlog.

Communication in Scrum is essential. Therefore, there are four significant meetings. During the *Planning Meeting* all three roles decide together which requirements from the Product Backlog should be processed during the next Sprint. They move all tasks to the Sprint Backlog. In the *Daily Scrum Meeting*, the team meets the Scrum Master for some short feedback. The Product Owner can participate. In the end of a Sprint there is the *Review Meeting* where the team shows the Product Owner what they have done. And there is also the *Retrospective Meeting* that acts as a feedback meeting for the team and the Scrum Master. In this meeting, they evaluate the last sprint and discuss what could be improved in the next sprint [8] [9] [10].

III. CYBERNETICS

According to Norbert Wiener, cybernetics is the science of control and the regulation of systems under real-time conditions. This also includes the automation and information processing of such systems. It is important to note regularities and to recognize the functional patterns of complex systems instead of specific details [3]. The real origin of cybernetics lies in nature and not - as often mistakenly assumed - in computer science. The most important factors of cybernetics are control, regulation and feedback. It can be referred to as a science of functioning [5].

This section provides a brief overview of the characteristics of complex systems and the fundamentals of cybernetics. At the end of this section, all the essential and relevant aspects for the comparison with the process model Scrum are summarized.

A. Characteristics of complex systems

In general, we have to distinguish between simple and complex systems. Simple systems are easily predictable. They are also easily applicable even without having knowledge of cybernetics. Complex systems can cause substantial problems if not held under control as they are much more interconnected and highly dynamic. It is not possible to intervene easily because this can result in unpredictable side effects [5].

The term "system" regarding cybernetics always means open systems, which interact with the environment and adapt to it over time. In contrast, closed systems do not interact and are self-contained. In such a given open and complex system, we cannot reduce complexity in order to simplify it, as it is often claimed. Complexity means variety. It is inevitable if the system has to accomplish all of its tasks reliably. If the complexity would be reduced, also variety would be reduced. Under certain conditions the system would fail. So we have to master complexity and to make use of it instead of eliminating it [2].

B. Importance of feedback

Due to cybernetics, "information" was recognized as third essential basic item supplementing the two basic elements "energy" and "matter", which both are not sufficient to explain how a system behaves. Information is the key which describes how things are organized in a dynamic system [2]. And to handle and manage such systems reliably, a special kind of information is needed, namely feedback. Without feedback it is simply impossible to hold a complex system under control [3].

C. Basic rules to control complex systems

Cybernetics presents the laws of nature which are responsible for the reliable functioning of complex systems. Relevant rules are mentioned below:

1) Circularity: Circularity is gained by regular feedback. A cybernetic system works because of control loops, circularity and feedback. That means that the system gives itself feedback. Through that process, the systems can excuse errors and are very robust [7]. The technical representation of this aspect is the well-known feedback control loop in Figure 2. It uses the cybernetic terminology and shows that the regulator compensates the influences of any disturbance keeping the system under control. Such systems follow an evolutionary approach. They are self-adjusting. And this is why such systems are viable in the long-term.

2) Self-organization and self-regulation: The mentioned self-adjusting aspect includes also self-organization and self-regulation. A cybernetic system is not externally directed. Instead, it is autonomous in the context of the whole system. It directs and controls itself in order to cope with complexity.

 ISBN: 978-1-61208-367-4

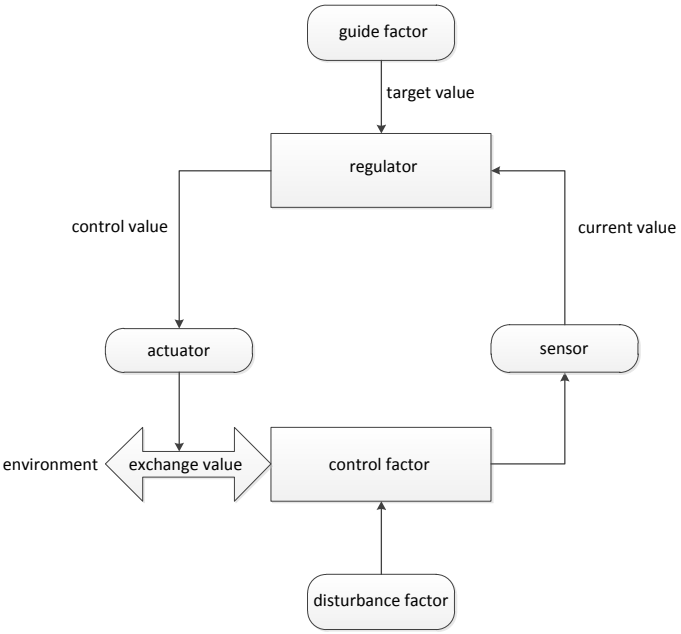


Figure 2. Feedback control loop according to Vester [7].

3) Theory of recursive systems: Cybernetic systems are long-term viable if they consist of interconnected self-adjusting subsystems building a recursive structure. The single systems interact with each other to represent the whole system.

4) Law of requisite variety: If we want to control a complex system, we need at least as much complexity and variety in the control mechanism as the controlled system itself has. If there is too less variety in the control system then the controlled system will sooner or later get out of control, as under certain conditions the control system cannot react properly to dominate the situation. The British cyberneticist and neurophysiologist W. Ross Ashby discovered the law of requisite variety, also called "Ashby's Law" [11]:"Only Variety can destroy Variety". It is the basic law of complexity. In simple terms, consider a switch with the two possible states "0" and "1". It is obvious that it is not able to control a system having three or more states. In order to control a system, we need as much variety (which means complexity) as the system itself [2].

D. Patterns and interconnections instead of details

To understand a complex system, it is not necessary to know as many details as possible. An abstract consideration and the recognition of patterns is the right way to understand a system. Interconnected thinking is extremely important. It is not the details that are important in a system but the coherences. The interconnections between the individual parts are important.

E. Cybernetic aspects put together

To summarize, complexity cannot be reduced but dominated. It is important that we work with it instead of fighting against it. To master complexity we need:

- self-regulation and self-organization,
- circularity for repeated and continuous operation,
- regular feedback in real-time for deterministic adjustments,
- communication and interconnectedness for a proper and continuous flow of information,
- autonomy allowing self-organization and self-control, and an
- evolutionary approach for possible adaptation due to changing conditions in the environment over time.

IV. CYBERNETICS AND SCRUM

Having shown the fundamental concepts, we will now compare Scrum with the essentials of cybernetics to verify whether it supports all the requirements of a sustainable and functional process. This includes mastering complexity, circularity, feedback, communication, real-time aspects, interconnectedness, autonomy, evolutionary approach, self-organizing and self-adjustment.

Circularity: The process model Scrum works in iterations. Scrum is already cyclically arranged since Sprints recur after a certain amount of time. During the Review Meeting the Product Backlog can be filled with new requirements from the customer as well. The result of the Retrospective Meeting is that the team can work better than in the Sprint before because they have reviewed the problems and obstacles. All the meetings improve the effectiveness of the team and the product quality. Therefore, the behaviour of the next Sprint is positively influenced. The iterations in Scrum represent the circularity required by cybernetics. It is the basis for continuous adaption and optimization and forms the control loop to keep the system under control.

Feedback in real-time: Feedback is established in various ways in Scrum. Considering the Sprint, which is determined by a duration of up to four weeks, there is the Sprint Backlog, which should be executed in order to get the product increment. At the same time, the Scrum Master gets the feedback of the team from a technical and a personal point of view out of the Review and Retrospective Meeting. Equally important is the feedback from the Product Owner and the customer which plays a minor role in Scrum. It is essential in order to detect problems early and to reach the goal in time. The Daily Scrum Meeting can also be seen as a feedback loop. This is real-time feedback. The Scrum Master can immediately react and appropriately guide the project if a team member is blocked by an obstacle and not able to work. Also, the team gets feedback from the outside world representing an open system.

Communication: Communication occurs between all the roles. In the Daily Scrum, regular communication takes place every single working day and also during all the other meetings. Through this constant communication, decision making processes improve. The purpose is that information flows between all stake holders. This is achieved in Scrum.

Interconnectedness: All the roles in a Scrum project are working together. All are interconnected, and this leads to a simplified coordination and a higher product quality as there are short communication paths. This is very important for direct information exchange.

 ISBN: 978-1-61208-367-4

Autonomy: The guidelines of Scrum let all team members work independently. They are free to act and decide during a sprint. This allows the team to do the right things right, as they are the experts in their specific domain. Autonomy also happens at the beginning of a Sprint when the development time of each work package has to be estimated. Every member estimates his task for the next sprint autonomously, which ensures much higher accuracy. Autonomy comes through the self-organizing teams, which means that they bear a great responsibility to work in a disciplined manner. Despite the autonomy, Scrum functions as a superior guideline.

Evolutionary approach: An evolutionary approach is inherent in every cybernetic system. It ensures a continuous development and adaption to changing conditions surrounding the system. So, the evolutionary approach keeps the cybernetic system viable. Looking at Scrum, it shows that changes of the agile process model are basically possible. Change proposals can be submitted. But one has to bear in mind that it is at the sole discretion of Jeff Sutherland and Ken Schwaber to apply any changes to the process model itself. Strictly speaking, this follows not the idea behind an evolutionary approach. Cybernetic systems are free to adapt to changing conditions as soon as they appear. Such systems do not have to wait for anything or anyone and are in an continuous process of adaption. In contrast, Scrum argues that it changes its framework infrequently.

From the strict cybernetic point of view, the process itself is not designed to change or adapt itself according new requirements appearing from the outside world. This is a missing aspect in scrum that cybernetic systems must inherently have. It means that the process model will support today's projects but unless adaptions it is uncertain if it will fit in future projects.

Self-organizing and self-adjusting: For the members of the Scrum Team, there is no precise formal rule or guideline how they have to do their job. Therefore, they can freely adapt to unknown complex project in the required way. They adjust independently to the task to be solved. That means that Scrum Teams regulate and organize themselves. So, they are working more efficiently, more motivated and more effectively because they have no precise rules to which they must adhere. Due to all these characteristics, the productivity can be sustainably increased.

So far we see that Scrum serves well as a lightweight agile process model, which helps to cope with the development of products with a complex scope. The structure of the agile process model combined with the distribution of the different roles overcomes the challenge of complexity and delivers a satisfactory result. If it comes to mastering complexity, everything collaborates including iterations, communication and regular feedback, self-organization and self-regulation. As Scrum supports todays projects well, we identified a lack in Scrum itself. It is not designed to adapt itself to new requirements.

V. BIOCYBERNETICS AND SCRUM

Cybernetics has its origin in biocybernetics. As a natural science, biocybernetics represents the fundamentals of the way living systems and organisms in the nature function successfully. Most natural systems have to deal with substantially more complexity than any technical system made by humans. Since billions of years, nature functions reliable and most efficient [12]. Frederic Vester, the founder of biocybernetics, defined eight basic rules every complex biological system has to fulfill in order to survive. Simply, these rules are principles of nature [7]. They can guarantee a successful evolutionary existence as they present possibilities for long-lasting development and survival of any living system - if it is a human being, an ecological system, a company or a city [13]. So biocybernetics represents the natural basis of cybernetics, which means that cybernetics comprises the corresponding science. Therefore, every technical system or process has to fulfill this eight rules of biocybernetics to be long-term viable.

As this is rather important, research has been done to analyse the biocybernetic aspects of Scrum. As an in-depth discussion of biocybernetics goes far beyond the scope of this paper, we give a short introduction to each of the eight rules and present our results afterwards. We recommend Vester's book, *The Art of Interconnected Thinking*, [7] for a comprehensive discussion of biocybernetics.

1st rule: Negative feedback cycles must dominate over positive feedback: In cybernetics, this means that for a system it is very important to be stable against interfering influences. To explain the principle briefly, an example for negative feedback is the control loop of the thermostat of a heating system. If a certain temperature is reached the energy input has to be decreased in order to not exceed the temperature value. If the temperature is too low, the energy input would be elevated. Negative feedback can also be found in nature. In general, there is almost exclusively negative feedback in order to keep the system stable. Positive feedback in nature appears in avalanches or steppe fire, for example. They build up continuously and achieve a new order at the end, which is exactly not desired in stable systems. That is why negative feedback has to dominate positive feedback, otherwise the system would collapse.

Negative feedback is therefore based on a control or feedback loop balancing the system. In terms of the Scrum model, this feedback control system means that the control factor would be the project and the regulator can be seen as Product Owner and Scrum Master. The guide factor leading the way would be the customer together with the market who will specify the product requirements. The team represents the actuator, which adjusts the actual and the desired condition. The current value is reported through direct communication and through test results of the product increment. Misconceptions can be critical for the process. They are reflected by the disturbance factor in the control loop. Misunderstandings can arise between the individual developers or between the team and the Product Owner or the Scrum Master. Also, longer absences due to illness or information deficits and wrong assumptions or misinterpretation of the requirements can lead to problems, which are solved through communication and correcting feedback during the sprint. And the feedback of the team at the end of a sprint leads to improvements that will be implemented in the next sprint.

In Scrum, the negative feedback mechanism is represented by this inherent control loop. It reliably prevents the process

 ISBN: 978-1-61208-367-4

to run out of control.

2nd rule: The function of the system has to be independent of quantitative growth: A system passes through metamorphoses while growing in order to survive. Based on the self-organization of cybernetic systems it does not have to be dependent on growth. Instead, there must be a restructuring step during the growth in order to move from one stable state to the next. After growth, the system is ready to get into the next phase. Therefore, the growth resembles a sustained "S"-shaped curve. But if a system suffers from unrestricted linear growth without proceeding to a stable state, it will lead to a collapse in the end. A butterfly would be the best example for growth and metamorphoses. The butterfly caterpillar pupates after a certain growth and envelopes itself in its cocoon to become a butterfly. At the hands of this transformation the linear growth is stopped, and continues as a butterfly in the next phase [7]. For complex systems, it can be deduced that reconstruction and metamorphoses are not replaceable by pure growth.

At this point the size of a Scrum Team matters. If a project gets bigger and more than five to nine people must work in a team, this team may not grow linearly. The project has to be split in order to get smaller teams that can work more effectively. This split is supported by the process model as Scrum is scalable for larger development teams. It is called Scrum-of-Scrums. It is important that the new sub-groups have superior coordination. There is a so-called Scrum-of-Scrums Meeting where all Scrum Masters of the single teams come together and can take over coordination [14]. If teams would grow linearly, communication would be very difficult because the communication channels rise exponentially with the number of people. The organizational overhead scales up and effectiveness and efficiency degrades. Finally, the system will run out of control and end in chaos. Therefore, linear growth should always be avoided. It needs "metamorphoses" for the purpose of a division in sub-projects.

3rd rule: The system must operate in a function-oriented, not product-oriented manner: The environment is constantly changing and that is why product requirements also vary. All products have a certain life cycle and will sooner or later disappear. On the other hand, the basic needs will not disappear and remain existent. The functional requirements of a product usually remain for a long time whereas products themselves change very often. For example, horse-drawn carriages are substituted by cars, telephones by modern smartphones, while mobility and communication as basic human needs further exist. Hence, it is always important to think function-oriented.

Scrum works without dedicated products. It works regardless of whether a database application, a smartphone application, a server application or a desktop application is the product. Scrum is detached from products and represents the "function".

4th rule: Exploiting existing forces (Jiu-Jitsu-Principle): Normally, Jiu-Jitsu is a Japanese martial art, which is used for a self-defence. The main principle in Jiu-Jitsu is that the force of the opponent should be utilized instead of defending against it. In systems this energy serves as control-energy. Applied here it means that the existing force and energy should be used instead of rejecting it.

This rule can be reflected in Scrum. Scrum uses the customer and the market with its requirements and wishes as external energy. They call the shots and decide what will happen next, so they are the driving force for the project. All changes that they bring along are very important for the quality of the final product and their competitiveness, as already the Agile Manifesto states [15]: "Welcome changing requirements".

5th rule: Multiple use of products, functions, and organizational structures: Viable systems put emphasis on reusability. If every product would be designed, produced, sold and used on its own, the costs and energy input would increase considerably. The efforts of energy, matter and information can be reduced substantially gaining a synergy effect.

In every project guided by Scrum, the organizational structure is intentionally left identical. Meetings are at the same time, at the same place, and have the same structure. This leads to more efficiency as stable conditions makes it a routine work. Another aspect is the specific knowledge of the Scrum Team. This knowledge and also existing software products, like libraries or frameworks, can be used in other projects. This makes multiple use of products and functions very easy and comfortable in Scrum.

6th rule: Recycling: Nature never produces garbage. Due to its cyclic and interconnected processes, waste does not exist and is used elsewhere as important resource. Recycling is one of the most important rules humans should revert to in order to keep a system alive.

It is hard to incorporate this rule into the process model Scrum because a Scrum Team will not produce waste in terms of material waste. Maybe, functions which are implemented in the actual Sprint and will not be used in the end can be seen as trash. But they do not have to be recycled. They can, for example, be provided for other teams or taken as features. The development is usually consumer-market-controlled so there is no real overproduction.

7th rule: Symbiosis: This usually means the cohabitation of two or more species in a common environment, which benefit from each other. In order to enable symbiosis, diversity in a small space is required. That means that many different elements within a system can share resources and functions in order to help other elements accomplishing their work in a more effective way.

The members of a Scrum Team should be located in close proximity to each other. This enables better communication and cohabitation through symbiosis. They benefit from each other if there are any problems. Here, especially the Scrum Master takes action and eliminates any obstacles or issues so that the team can work efficiently. And arising requirements not discussed so far can be cleared by the Product Owner, which itself benefits from much higher product quality. They complement each other.

8th rule: Biological Design: All systems, products, functions or organizations should be developed in respect of the nature. Building anything against nature is plain unnatural. So, nature always matters as it defines what is right or wrong.

 ISBN: 978-1-61208-367-4

Scrum has to correspond to the structure of a viable system and may not be unnatural in its structure and process. This sounds abstract but means nothing else than Scrum has to follow natural processes if it wants to act viably. Scrum fulfills this requirement of rule eight as it follows biological design and not an artificial one. It brings along all these preconditions of biological design with circularity, feedback, autonomy, self-organization, recursive structure and all other mentioned aspects satisfying this last rule of biocybernetics.

In summary, the fundamental eight biocybernetic rules are met in different degrees. From the perspective of our research activity, basically most of them can be seen as fulfilled. Looking at the third rule where functions dominate products, shows that the point of view is essential: as Scrum is not focused on special products and therefore flexible for software projects and applications of different kinds, it fulfills this rule quite good. In contrast, if we look at Scrum itself, it shows some shortcomings in continuous adaption to changing conditions. The sixth rule, which means recycling to avoid waste, is applicable only partly due to the immaterial nature of computer science.

VI. Conclusion and Future Work

Even if Scrum fulfills most of the requirements cybernetic systems must have, we come to the conclusion that it is not a true cybernetic process. Not only that Scrum does not claim itself to be cybernetic. The history of Scrum begins in lean management strategies of Japanese companies. It incorporates a lot of best practices and has not been designed with cybernetics in mind.

Nevertheless, Scrum is of course very suitable for today's software development projects. As responding to change is an important aspect in Scrum, it addresses a fundamental concept of cybernetics to hold a system under control. And there are quite more major principles in Scrum beside circularity and feedback, namely autonomy, self-organization and self-adjustment within the context of the overall process structure.

Scrum guides the project management process in the right way and successful projects are no accident. Also the biocybernetic requirements are largely fulfilled, which leads to the same conclusion. Although, some of the rules are not directly applicable due to the immaterial speciality of Computer Science, we consider them as satisfied, as we have not discovered major inconsistencies or conflicts.

If we do not look at today's projects but on projects in the distant, or maybe, not so distant future, the missing evolutionary approach must be mentioned. Submitting change proposals differs from a cybernetic way. But at this point, Kanban can be deployed [16]. Kanban is a management technique for software development incorporating continuous improvement of the process itself in small steps. So with Scrum and Kanban combined, this essential cybernetic aspect can also be satisfied, which keeps the system long-term viable.

Beside the overall process, which has been analysed here, shortcomings can be discovered in some other areas. For example, Scrum defines "roles" although cybernetics requires "functions". The process model uses this term in order to determine key tasks and to define responsibilities. So, every role has a certain focus as already mentioned, but Scrum does not explicitly forbid additional tasks arising during the development process. In practice, it is often seen that additional tasks are carried out in order to get a product with the required quality. Therefore, autonomy and self-organization are the key aspects to get this done right, although this is not noted in Scrum.

Another issue concerning autonomy and self-organization is the Scrum Team. Scrum does not define any cybernetic approach the team has to follow. Therefore, it can be completely ignored meaning that the recursive cybernetic structure is broken. As before, it is the responsibility of the autonomous team to do the things right. The prerequisites are met as both, Scrum Master and Product Owner, can be present during the Daily Scrum Meeting in order to support a cybernetic approach.

After this analysis of the coherences between Scrum and cybernetics it can be seen that many cybernetic aspects are already covered in Scrum. So far, our recommendation is to additionally apply Kanban and basic cybernetic principles in order to overcome the mentioned shortcomings. In future work, we will analyse this promising combination in more detail.

References

[1] J. Sutherland and K. Schwaber. The Scrum Guide. [Online]. Available: https://www.scrum.org/Portals/0/Documents/Scrum Guides/2013/Scrum-Guide.pdf (2014.05.23)

[2] F. Malik, Komplexität - was ist das?(Complexity - What is this?). Cwarel Isaf Institute, 1998.

[3] N. Wiener, Cybernetics: or Control and Communication in the Animal and the Machine. New York: Wiley, 1961.

[4] F. Malik, Strategy: Navigating the Complexity of the New World. Campus, 2013.

[5] ——, Strategie des Managements komplexer Systeme: Ein Beitrag zur Management-Kybernetik evolutionärer Systeme (Management Strategy of Complex Systems: A Contribution of Management Cybernetics of evolutionary Systems). Haupt, 2008.

[6] ——, "The six cultural values for the reliable functioning of an organization," Malik on Management, Volume 22, 2014.

[7] F. Vester, The Art of interconnected thinking. Mcb, 2013.

[8] A. Cockburn, Agile Software Development. Addison Wesley, 2006.

[9] H. Wolf and A. Rock, Agile Softwareentwicklung: Ein Überblick (Agile Software Development: An Overview). dpunkt.verlag, 2011.

[10] K. Schwaber, Agile Project Management with Scrum. Microsoft Press, 2004.

[11] R. Ashby, An introduction to cybernetics. Chapman Hall, 1956.

[12] F. Vester, "The biocybernetic approach as a basis for planning our environment," System Practice, Volume 1, No. 4, 1988.

[13] G. R. M. Harrer, "The biocybernetic approach as a basis for planning and governance," Proceedings of the 54th Meeting of the International Society for Systems Sciences (ISSS), Waterloo, Canada, 2010.

[14] U. Biberger, Gestaltungshinweise für agile Software-Entwicklungsprojekte unter dem Blickwinkel der Kybernetik (Design Guidelines for agile Software-Development Projects from the Viewpoint of Cybernetics), 2009.

[15] Agile Alliance. Manifesto for agile software development. [Online]. Available: http://agilemanifesto.org/principles.html (2014.05.23)

[16] D. Anderson, Kanban. Blue Hole Press, 2010.

 ISBN: 978-1-61208-367-4

Can Functional Size Measures Improve Effort Estimation in SCRUM?

Valentina Lenarduzzi
Dipartimento di Scienze Teoriche e Applicate
Università degli Studi dell'Insubria
Varese, Italy
valentina.lenarduzzi@gmail.com

Davide Taibi
Software Engineering Research Group
University of Kaiserslautern
Kaiserslautern, Germany
taibi@cs.uni-kl.de

***Abstract*—In SCRUM projects, effort estimations are carried out at the beginning of each sprint, usually based on story points. The usage of functional size measures, specifically selected for the type of application and development conditions, is expected to allow for more accurate effort estimates. The goal of the work presented here is to verify this hypothesis, based on experimental data. The association of story measures to actual effort and the accuracy of the resulting effort model was evaluated. The study shows that developers' estimation is more accurate than those based on functional measurement. In conclusion, our study shows that, easy to collect functional measures do not help developers in improving the accuracy of the effort estimation in Moonlight SCRUM.**

Keywords: Software Effort Estimation, Agile Development, SCRUM effort estimation, Functional measurement.

I. INTRODUCTION

Agile methodologies call for different and possibly more complex effort estimation techniques than other methodologies [10]. This is due to the iterative nature of projects that use agile methods and the lack of detailed requirements and specifications at the beginning of the project.

Several effort estimation models have been defined based on user experience or on previous project results but, due to the differences between different development methodologies, the applicability of those estimation models appears to be limited.

In this work, we focus on SCRUM [13] as reference process (see Figure 1).

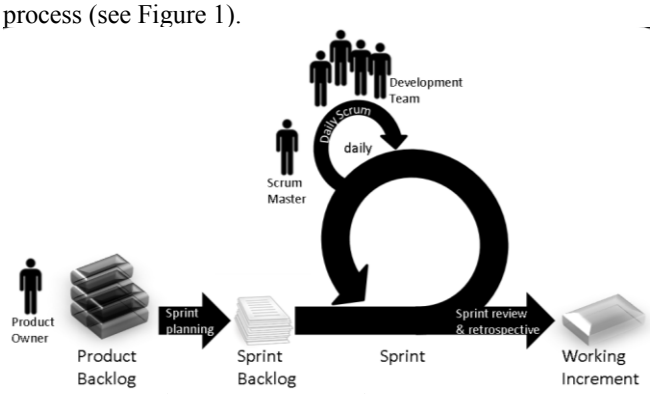


Figure 1: SCRUM Development Process

Requirements in SCRUM are collected in the "product backlog" and described as "user stories".

During the Sprint Planning Meeting, the team estimates the effort for the user stories in the product backlog based on their experience on implementing similar user stories. Then, they predict the amount of user stories they believe can develop in the upcoming sprint. The consequence is that teams need to adjust their project plan, during each sprint meeting.

SCRUM does not prescribe a unit of measure to estimate the effort. Common estimating methods include numeric sizing, t-shirt sizes, and story points.

In this work, we investigate if it is possible to use functional measures to help developers increase the accuracy of the effort estimation in SCRUM.

For this reason, we conducted an empirical study on a SCRUM project developed with Moonlighting SCRUM [7], a version of SCRUM slightly adapted for part-time developers working in non-overlapping hours.

The remainder of this paper is organized as follows: Section 2 describes related work. Section 3 presents the context where we applied this study. Section 4 first introduces the research questions and derive goals and hypotheses, then elaborates on the measurement instruments and study design. Section 5 presents the results of the study. Section 6 describes the threats to validity and finally Section 7 draws conclusions and gives an outlook on future work

II. RELATED WORKS

Several empirical studies report that developers usually underestimate their effort in agile processes, compared to other methodologies [10]. Other studies analyzed the accuracy of the effort planned and spent for implementing user stories, reporting overoptimistic and sometimes unrealistic initial estimates [4][11]. Moreover, a case study run by Chao also reported that the effort estimation does not improve over time [4].

One of the first attempts to help developers improve the estimation in SCRUM has been published by Jamieson in 2005 [1]. Jamieson identified a set of estimation problems in SCRUM such as the need of budget reallocation due to the requirement volatility resulting in heavy and costly change management.

Lavazza [8] identified a set of potential problems such as the different nature of the user stories, the size of a sprint and velocity. Moreover, he also highlighted the importance of choosing the correct granularity level for measures and collect historical data.

Buglione et al. [6] proposed to apply functional size measurement methods in a late stage of the process, when requirements become available and are more stable.

 ISBN: 978-1-61208-367-4

Ziauddin et al. [14] propose an early estimation model for SCRUM based on historical data. They calculate the effort based on the number of user stories, the team velocity, the sprint duration adjusting the results based on a set of influencing factors such as the team composition, environmental factors and team dynamics. The model has been calibrated on 21 SCRUM projects and provides a good accuracy. However, the model is only suitable for projects where the requirements are clear and fixed at the beginning of the project.

Fuqua [16] ran a controlled experiment with the goal of understanding if functional measurement in XP-Projects can help to produce a more accurate schedule, and if functional measurement can help to predict how long it will take to implement a story. Results show that Function Points (FP) are unable to estimate the required effort. Moreover, FP have a too fine granularity and require sizeable measurement effort due to the complexity of the FP measurement process.

Finally, a recent work published by Popli and Chauhan [12], proposes to use a new unit of measure: the "sprint points". Sprint Points are calculated combining information related to the project type, requirement quality, hardware and software requirements, requirements complexity, data transactions and number of development sites.

III. CONTEXT

In this section, we describe the development process we analyzed in our study and the application that was developed.

This work is based on the development of Process Configuration Framework (PCF), an online tool to classify software technologies and identify tool chains in specific domains [15]. PCF is a relatively small application, composed of 12,500 effective lines of code, calculated without considering comment lines, empty lines, and lines containing only brackets. The development started in February 2013, based on an existing prototype, and the first version of the tool was released at the end of May 2013.

PCF is developed in C#/Asp.net with a simple 3-tier architecture that allows the development of independent features among developers. This allows developers to work independently on the data layer, on the business layer and on the presentation layer.

We deal with a special case of SCRUM process. In fact, special development conditions called for some changes of the SCRUM process.

The development was carried out by four part-time developers (Master's students) with 2 to 3 years' experience in software development. Developers work in non-overlapping hours and, to manage a good level of communication, an online forum is used for the daily meeting, as prescribed by Moonlight SCRUM [7]. Moreover, sprint retrospectives, planning, and retrospective discussions are led by means of an online integrated tool (http://www.rallydev.com), which allows us to record sprint reports, manage product backlog, and draw burn-down charts.

The development process was organized as follows.

a) The duration of each sprint is three weeks

b) Daily meeting are replaced by reporting on an online forum twice a week

c) A user story can be assigned only to a single developer

d) Every developer works in isolation.

The work is coordinated by the SCRUM master via the weekly meetings.

IV. THE CASE STUDY

We formulate the goal for our study following the Goal Question Metric approach [5] as:

analyze the development process *for the purpose of* evaluating the effectiveness of estimation measures *from the viewpoint of* the developers *in the context of* a *moonlight* SCRUM development process

A. Metrics

Since measures are collected to estimate effort, a characteristic of these measures is that they can be measured before development. So, in principle we expect that it is possible to build a model that, by linking the development effort to the measures, provides an estimation tool that can be used in conjunction with (and possibly even in place of) the usual agile estimation techniques.

Another characteristic of the measures is that they must be fast and easy to collect, since they have to fit in an agile process, where little time and effort can be dedicated to measurement activities. Moreover, the proposed measures are easy to collect, so that any developer can perform the measurement without problems.

To measure user stories, we considered the usage of traditional functional size measures, possibly adapted to the agile context. However, plain function points such as IFPUG (International Function Point User Group)[18] or COSMIC function point [19] measures could not be used. In fact, we noticed several problems, including the following:

- The most popular functional size measures use processes (Elementary process or Functional process) as the element to be measured. This is reasonable when the smallest development step (for instance, a sprint in a regular SCRUM process, or an iteration in a RUP process) addresses several processes.
  However, in our case the development of a single process could span multiple sprints. Accordingly, knowing the size of a process could hardly help estimate the work to be done in a single sprint.
- Several sprints involved working mainly on the Graphical User Interface (GUI) of the application. So, functional size measures would not help estimate the effort required.
- Implementation-level details (like the number of interactions with the server or the number of database tables involved in the operations) appeared to affect the required effort.

Based on the aforementioned constraints, we defined the following measures to be collected during the planning game:

- Actual effort: number of hours spent per user story. This information is tracked by developers and collected at the beginning of each spring.
- Story Type: we collect this information so as to classify the user stories based on the type of development.
  - New feature: user stories that involve the creation of a new feature.
  - Maintenance: bug fixing or requirement changes for an existing feature.
- *Functional measures.* Since standard Function Points such as IFPUG or FISMA require a lot of effort to be collected, and most of required information is not available in our context, we opt for the Simplified Function Points (SiFP) [17]

SiFP are calculated as SiFP= 7 * #DF + 4.6 * #TF where #DF is the number of data function (also known as logic data file) and #TF is the number of elementary processes (also known as transactions).

We collect SiFP instead of IFPUG Function Points, since SiFP provides an "agile" and simplified measure, compatible with IFPUG Function Points [17].

Moreover, before running this study, we asked our developers what information they take into account when estimating a user story. All developers answered that they consider four pieces of information, based on the complexity of implementing the GUI and the number of functionalities to be implemented. They usually consider each GUI component as a single functionality that requires the sending or receiving of the information to the database. The complexity of the communication is related to the number of tables involved in the SQL query.

For these reasons, we also consider the following measures:

- GUI Impact: *null, low, medium, high:* complexity of the GUI implementation identified by the developers.
- # GUI components added: number of data fields added (eg. Html input fields)
- # GUI components modified: number of data fields modified
- # database tables: number of database table used in the sql query.

We can consider this last measure as a functional size measurement with a very low level of granularity, even though not directly comparable to SiFP or IFPUG Function Points.

B. Study Procedure

The measures identified are collected during each sprint meeting by the SCRUM master, in an Excel spreadsheet.

After each sprint we collect the actual effort spent for each story, in order to validate results.

Measures must be collected in a maximum of 5 minutes per user story, at the end of the usual SCRUM planning game, so as to not influence the normal execution of the required SCRUM practices.

Developers were informed, through an informed consent that the information is collected for research purposes and will never be used to evaluate them.

V. RESULTS

We ran the study analyzing the data for 4 months. We ran 6 sprints of three weeks each with 4 developers working part-time for the entire period.

Table I reports descriptive statistics on the user stories per story type. As shown in this table, the vast majority of the user stories are related to the development of new features (65%) while only 35% on maintenance.

Considering GUI impact (Table II), we can see that most of the user stories are related to the development of graphical features with high or medium complexity.

Functional measures have been collected only for 55 user stories (40.4%) since the remaining user stories do not contain enough information for functional size measurement (e.g., GUI features do not deal with data transactions).

As expected, the number of GUI components added or modified increase paired with the GUI impact while unexpectedly, the higher the GUI impact, the lower is the number of hours required for implementing a user story.

TABLE I. ACTUAL EFFORT PER STORY TYPE

		All	New Feature	Maintenance
# User stories		136	99(73%)	37 (27%)
Effort per user story (hours)	Avg	3.16	3.68	1.96
	Median	2.00	2.00	2.00
	Std. Dev	2.91	3.28	1.01

TABLE II. EFFORT AND GUI COMPONENT ADDED OR MODIFIED (GUI_COMPONENTS) PER USER STORY PER GUI IMPACT

GUI Impact		Story Type		
		All	New Feature	Maintenance
Null	#User Stories	11	6	5
	AVG (hours)	3.12	1.91	1.6
	AVG (GUI_Comp)	5.27	3.67	0.2
Low	#User Stories	30	26	4
	AVG (hours)	3.68	2.46	1
	AVG (GUI_Comp)	1.33	1.44	1
Medium	#User Stories	40	30	10
	AVG (hours)	1.96	3.50	1.70
	AVG (GUI_Comp)	5.02	6.13	0
High	#User Stories	55	37	18
	AVG (hours)	1.30	4.90	2.20
	AVG (GUI_Comp)	8.28	7.89	9.05

 ISBN: 978-1-61208-367-4

Descriptive statistics for the SiFP collected for the user stories (see Table III) show that user stories with a null GUI Impact (user stories that do not deal with the user interface) have the higher number of SiFP, followed by the stories with a high GUI impact.

TABLE III. SIFP PER USER STORY PER GUI IMPACT

GUI Impact		Story Type		
		All	New Feature	Maint.
All	#User Stories	55	47	8
	AVG (SiFP)	6.1	5.76	8.58
Null	#User Stories	7	2	5
	AVG (SiFP)	9.12	6.4	12.51
Low	#User Stories	19	18	1
	AVG (SiFP)	4.66	4.8	2.2
Medium	#User Stories	22	20	2
	AVG (SiFP)	5.69	6.06	1.96
High	#User Stories	7	7	0
	AVG (SiFP)	8.79	8.79	/

After the analysis of descriptive statistics, we investigated the correlations from actual effort and:

- SiFP
- GUI components (Added + Modified)
- GUI components added, modified and database tables

Here, we report the results for all user stories and for each GUI impact and story type, so as to understand if this information can improve the estimation accuracy.

The analysis of correlations among SiFP and effort reported in all user stories does not provide any statistical significant result (Table IV – column "All Projects" and Figure 2), showing a very low goodness of fit (MMRE=81.4%, MdMRE=135.3%).

The analysis was then carried out by clustering stories per story types and GUI impact. Results obtained after the clustering show the same behavior, except for stories implementing new features with a low GUI impact (Table IV – Column "GUI Impact Low – Features"). In this case, results are statistically relevant but with a very low goodness of fits. (MMRE=147%, MdMRE=111%).

The correlation between the actual effort and the number of GUI components added or modified shows a similar pattern to the previous one in Table V and Figure 3. Only the analysis of stories with a medium GUI impact provides statistically significant results but, together with the analysis of the other types of stories, there is a very low correlation with a very low goodness of fit. (MMRE=71.3%, MdMRE=140.1%). Results are also confirmed by grouping user stories by story type and impact.

Finally, the multivariate correlations among GUI components added, modified and database tables provides statistically significant results paired with a low correlation. Moreover, also the multivariate correlation does not increase the goodness of fit (Table VI and Figure 4).

TABLE IV. CORRELATIONS AMONG EFFORT AND SIMPLIFIED FUNCTION POINTS

	All Projects	GUI Impact											
		Null			Low			Medium			High		
Story Type		All	Feat.	Maint.	All	Feat.	Maint.	All	Feat.	Maint.	All	Feat.	Maint.
#User Stories	55	7	2	5	19	18	1	22	20	2	7	7	0
pearson	0.065	0.391	/	0.383	0.660	0.669	0	-0.068	-0.073	/	-0.370	-0.370	0
p-value	0.320	0.193	/	0.262	0.001	0.001	0	0.382	0.380	/	0.207	0.207	0
R^2	0.004	0.153	/	0.147	0.436	0.448	0	0.005	0.005	/	0.137	0.137	0

TABLE V. CORRELATIONS AMONG EFFORT AND GUI COMPONENTS ADDED OR MODIFIED

	All Projects	GUI Impact											
		Null			Low			Medium			High		
Story Type		All	Feat.	Maint.	All	Feat.	Maint.	All	Feat.	Maint.	All	Feat.	Maint.
#User Stories	136	11	6	5	30	25	5	40	30	10	55	36	19
pearson	0.071	-0.138	0.146	-0.211	0.191	0.190	0	0.436	0.396	0.588	-0.196	-0.217	0.040
p-value	0.207	0.343	0.391	0.366	0.156	0.181	0	0.002	0.015	0.037	0.076	0.102	0.437
R^2	0.005	0.019	0.021	0.045	0.037	0.036	0	0.190	0.156	0.346	0.038	0.047	0.002

 ISBN: 978-1-61208-367-4

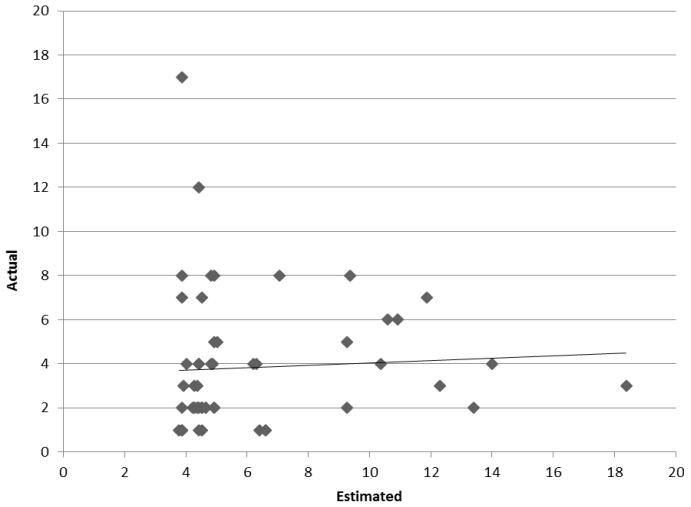


Figure 2: Actual Effort vs Estimated Effort with SiFP

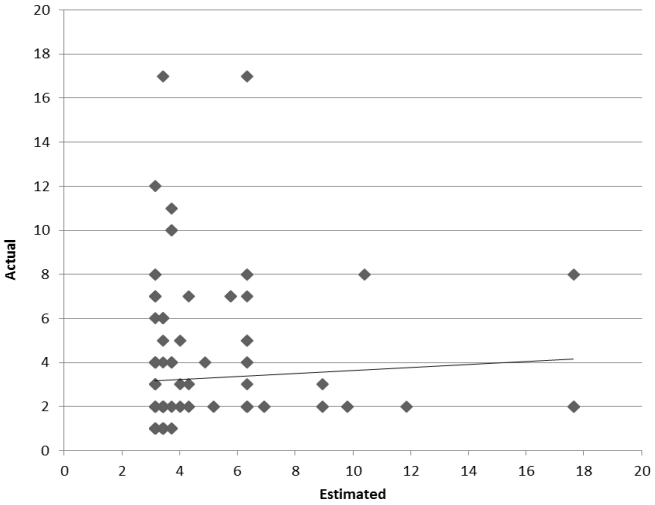


Figure 3: Actual Effort vs Estimated Effort with GUI components added + modified

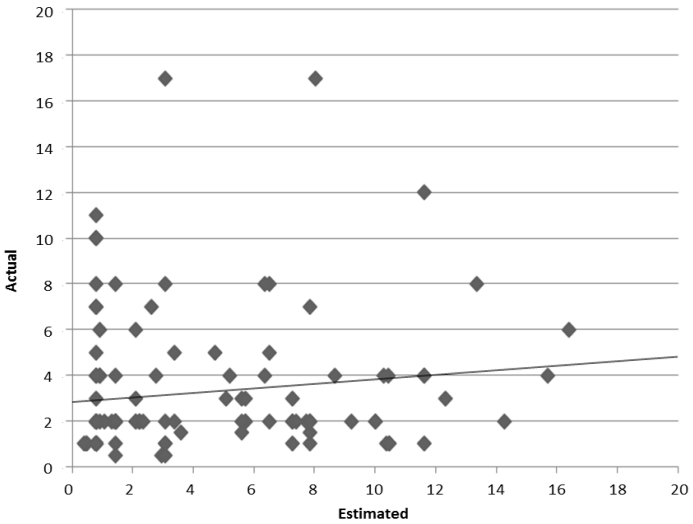


Figure 4: Actual Effort vs Estimated Effort with GUI components added, modified and database tables involved

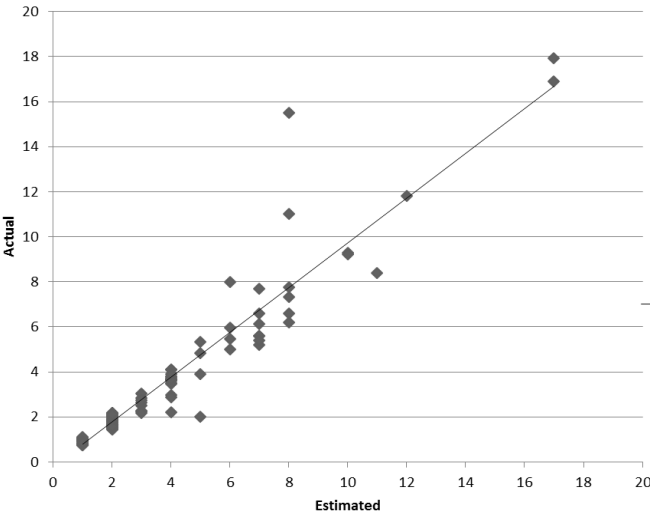


Figure 5: Actual Effort vs Developers' estimated effort

TABLE VI. MULTIVARIATE CORRELATION AMONG ACTUAL EFFORT AND GUI COMPOMENTS ADDED, MODIFIED AND DATABASE TABLES.

		GUI Comp Added	GUI Comp Modified	Database Tables
	#Projects	138	138	138
Pearson	**Actual Effort**	0.212	-0.033	0.130
	GUI Comp Added	1.000	0.272	0.391
	GUI Comp Modified	0.272	1.000	0.377
	Database Tables	0.391	0.377	1.000
p-value	**Actual Effort**	0.006	0.351	0.0064
	GUI Comp Added		0.001	0.000
	GUI Comp Modified	0.001		0.000
	Database Tables	0.000	0.000	
	R^2	0.061		

In order to understand if the results are due to errors in the effort estimation made by our developers, we finally analyze the accuracy of the effort estimation carried out by our developers. We compared the actual effort with the effort estimated before implementing the user story (see Figure 5). Results shows a very accurate estimation, with a very low average error (MMRE=13.5% MdMRE=9.35%). The low error is probably due to the nature of the user stories in Moonlight Scrum, usually smaller than common user stories in SCRUM. However, as expected, the accuracy decreases when the effort planned per user story is higher.

This confirms that in our project context, expert estimation is still much better than data driven estimation, based on functional measurement.

 ISBN: 978-1-61208-367-4

VI. DISCUSSION

The immediate result of this study is the low prediction power of functional size measures in SCRUM.

Unexpectedly, the prediction accuracy of SiFP compared to the accuracy of experience-based predictions is dramatically low.

Since SiFP can easily replace the more common IFPUG function points with a very low error [17], it appears that functional size measures are not suitable for predicting the effort in Moonlight Scrum.

Moreover, no correlations are found between the effort and the information commonly used by our developers to estimate user stories (GUI components and database tables). Again, the lack of correlation is probably due to the low complexity and the small effort needed to implement a story.

Results are based only on the analysis of one development process, based on a relatively small codebase (12500 effectives lines of code).

Concerning internal validity of the study, developers are master students, with a limited experience (2-3 years) in software development with at least one year of experience in SCRUM.

As for external validity, this study focuses on Moonlight SCRUM, a slightly modified version of SCRUM. We expect some variations in applying the same approach to a full time development team, working on a plain SCRUM process.

Regarding the reliability of this study, results are not dependent by subjects or by the application developed. We expect similar results for the replication of this study with a Moonlight SCRUM process.

VII. CONCLUSIONS

In this work, we analyzed the development of a Moonlight SCRUM process so as to understand if it is possible to introduce agile metrics to the SCRUM planning game.

With this study, we contribute to the body of knowledge by providing an empirical study on the identification of measures for Agile, and in particular SCRUM, effort estimation.

Therefore, we first gave an overview of the few existing empirical studies in the field agile and SCRUM effort estimation, then we introduced the context of this study and the case study we ran.

Results of our study show that SiFP do not help to improve the estimation accuracy in Moonlight SCRUM. Moreover, the accuracy does not increase considering other measures usually considered by our developers when they evaluate the effort required to develop a user story.

Since SiFP can easily replace the more common IFPUG function points with a very low error [17], we can conclude that, based on our case study, it appears that functional size measures are not suitable for predicting the effort in Moonlight Scrum.

Future work includes the replication of this study in an industrial context with a plain SCRUM process.

REFERENCES

[1] D. Jamieson, K. Vinsen, and G. Callender, “Agile Procurement to Support Agile Software Development”, Proceedings of the 35th IEEE International Conference on Industrial Informatics, pp. 419-424, 2005.

[2] T. Sulaiman, B. Barton, and T. Blackburn, “AgileEVM - Earned Value Management in SCRUM Projects”, Proceedings of AGILE Conference , pp. 10-16, 2006.

[3] N. C. Haugen, “An empirical study of using planning poker for user story estimation”, Proceedings of AGILE Conference , pp. 9-34, 2006.

[4] L. Cao. “Estimating Agile Software Project Effort: An Empirical Study” Americas Conference on Information Systems (AMCIS), paper 401, 2008

[5] V. R. Basili, G. Caldiera, and H. D. Rombach, “The goal question metric approach.” Encyclopedia of software engineering, pp. 528–532, 1994.

[6] L. Buglione and A. Abran. “Improving Estimations in Agile Projects: Issues and avenues” Proceedings of the 4th Software Measurement European Forum (SMEF) Rome (Italy), 2007

[7] D. Taibi, P. Diebold, and C. Lampasona. “Moonlighting SCRUM: An Agile Method for Distributed Teams with Part-Time Developers Working during Non-Overlapping Hours” Proceedings of the Eighth International Conference on Software Engineering (ICSEA), pp. 318-323, 2013

[8] L. Lavazza. “Managing Performance Impact Factors for Effort Estimation in Agile Projects”. PIFPRO'12 workshop. Collocated with IWSM/Mensura, 2012

[9] R. Meli, “Simple Function Point: a new Functional Size Measurement Method fully compliant with IFPUG 4.x”, Software Measurement European Forum, 2011

[10] B. Ramesh, L. Cao, and R.Baskerville. "Agile Requirements Engineering Practices and Challenges: An Empirical Study," Information Systems Journal. Vol. 20, Issue 5, pp 449–480, 2007.

[11] V. Mahnic. ‘A Case Study on Agile Estimating and Planning using SCRUM” Americas Conference on Information Systems (AMCIS), pp 123-128, 2008

[12] R. Popli and N. Chauhan. ”A Sprint-Point Based Estimation Technique In SCRUM” Information Systems and Computer Networks, pp.98-103, 2013

[13] K. Schwaber. “Agile Project Management with SCRUM” Microsoft Press, ISBN 9780735619937, 2004

[14] K. Z. Ziauddin, K. T. Shahid, and Z. Shahrukh. “An Effort Estimation Model for Agile Software Development” Advances in Computer Science and its Applications Journal. Vol. 2, No 1, pp 314-324, 2012

[15] P. Diebold, L. Dieudonné, and D. Taibi, “Process Configuration Framework Tool”, Euromicro Conference on Software Engineering and Advanced Applications , 2014.

[16] A. M. Fuqua. “Using function points in XP – considerations” International conference on Extreme programming and agile processes in software engineering, pp. 340-342, 2003

[17] L. Lavazza and R. Meli, “An Evaluation of Simple Function Point as a Replacement of IFPUG Function Point”, IWSM - Mensura 2014, Rotterdam, October 2014.

[18] International Function Point Users Group. “Function Point Counting Practices Manual”, 2004

[19] COSMIC - Common Software Measurement International Consortium. “The COSMIC Functional Size Measurement Method - version 3.0 Measurement Manual” September 2007

 ISBN: 978-1-61208-367-4

On Some Challenges in Assessing the Implementation of Agile Methods in a Multisite Environment

Harri Kaikkonen

Department of Industrial Engineering and Management
University of Oulu
Oulu, Finland
harri.kaikkonen@oulu.fi

Pilar Rodríguez, Pasi Kuvaja

Department of Information Processing Sciences
University of Oulu
Oulu, Finland
{pilar.rodriguez, pasi.kuvaja} @oulu.fi

***Abstract*—Organizations utilize agile development methods and multisite environments with the intent to reduce costs and development time. Assessing the results of utilizing and adopting such methods is also frequent. An assessment survey instrument was used to analyze the transformation of a multisite software development organization from waterfall-type development into agile development. The transformation was done in two globally distributed sites in Finland and India around 12 months apart. The assessment survey was conducted in the Finnish site 6 months after it had changed its working methods and again 12 months later in both sites. The site in India had adopted similar methods after the previous assessment survey was conducted. The results of the assessment survey in the Finnish site indicated regression between the two assessment rounds, while the results in India appeared to be better compared to Finland in the second round. Analysis of the results suggests that cultural differences and time elapsed from the organizational transformation may have influence in the assessment results and should be taken into account when assessing the implementation of development methods.**

Keywords-organizational change; global software development; agile methods; Scrum; process assessment.

I. INTRODUCTION

Adopting agile development methods like Scrum [1] and extreme programming (XP) [2] have seen a great deal of interest in the software development community because of their intended benefits of delivering working software and being more responsive to changes, among other reasons [3]. However, scaling agile methods into larger organizations than a single or a few teams has its difficulties and there have been several descriptions of how to do that (e.g., [4][5]).

As development organizations become larger, they are often also spread out globally out of necessity or because of their business environments [6], which causes a whole other array of issues to be considered in managing development work.

This publication describes selected results of a quantitative process assessment conducted at a medium-sized software development organization. The organization adopted a Scrum-based software development process in their multi-site organization. The adoption and the assessment were done in two phases. First, the process was adopted by a smaller unit in Finland with approximately 30 people, who were assessed approximately six months after the adoption. Then, with the experience gathered from the first site, similar processes were adopted in the same organization's site of about 50 people in India and the assessment was repeated in both sites. The adoption was also planned to be further expanded to other sites.

The aim of this publication is to provide evidence of issues in assessing the implementation of organizational changes such as new development processes in a global software development (GSD), or other multisite organization.

The remainder of the publication is organized as follows. Section II contains a description of related work as theory of agile development methods and global software development. Section III presents a description of the assessment process and Section IV a description of the organization in which the assessment was conducted. Section V presents the relevant results of the assessments. Section VI includes discussion based on the results and the paper concludes in Section VII.

II. RELATED WORK

The agile movement gained publicity within the community during the 1990's, and was later epitomized in the agile manifesto, published in 2001 [3]. The manifesto was a collaborative agreement of what practitioners saw as the values and principles of agile software development. In addition to the actual manifesto, the authors also described twelve principles behind it. The twelve principles were agreed as common to the agile practitioners, although agile methods had already been described and were in use in many different settings. 'Agile methods' is an umbrella term for a wide different set of approaches (e.g., Scrum, XP and kanban [7]), that have challenged the traditional waterfall model of software development and introduced a more lightweight process of producing software. Key differences between agile methods and traditional software development include iterative development and promoting empowered teamwork. However, a common misinterpretation of agile software development is that agility is achieved with practices and tools, although the focus should be on being agile, instead of doing it [4].

 ISBN: 978-1-61208-367-4

During the same time that agile methods started to become increasingly prevalent in software development, globalization of high-technology businesses have increased the need for GSD. Software and its use as both products and services has become a competitive weapon which must be utilized efficiently to stay ahead in high-technology competition [6].

The challenges of GSD have been clear from the start and have been described in several sources (e.g., [8][9][10]). Issues range from strategic level issues like how to divide work between different sites, to more tactical level problems like how to arrange effective daily communication channels, to more complex systems like cultural differences and their effect on project and process management [6]. It is clear that many types of issues become apparent when dividing any kind of work globally, and with development work that often realizes inside developers' and designers' heads, the problems can be all the more difficult. Methods have also been proposed to reduce the effects of the challenges involved with GSD. These methods range from the use of maturity models [11] to suggested practices and techniques [10].

As organizations try to improve their processes and products, they often turn to assessments to get further understanding of their processes. Many of these assessments have also been conducted in global development environments (e.g., [12]). Similarly to the identified challenges with GSD, analyzing assessment results from GSD organizations may also contain challenges that are unknown. This is true for assessment results in any multisite organization, not just for GSD organizations.

III. RESEARCH METHOD

One of the challenging things in any organizational transformation towards a new way of working is how to assess the transition process and guide the next steps. This research was conducted using the *Lean and Agile Deployment Survey*, which is an assessment instrument developed by the University of Oulu in collaboration with industrial partners in the Cloud Software Program [13] in Finland. The instrument is specifically designed for enabling an effective transformation to a lean and agile way of working. The survey is based on a generic structure of three organizational levels; portfolio, program and project [5], and focuses on four main dimensions: organizational set-up, practices, outputs and culture/mindset. The survey was part of a larger effort that University of Oulu was performing in identifying the right agile practices to adopt and to determine whether organizations are ready for lean and agile. Additionally, the approach is meant to provide information for deciding what necessary preparations and potential difficulties may be faced during the adoption process.

The conducted survey contained four context information questions for analyzing purposes, and over 70 statements that described the organization's agile development process as it had been planned and taken in use internally. The statements were tailored from general statements in the *Lean and Agile Deployment Survey* to correspond with the terminology and processes of the case organization. Some generic examples of the statements are presented below:

- The product backlog prioritization is clear
- The product owner guides the Scrum team by prioritizing the user stories
- I understand when the user stories are complete and can be accepted within the sprint

IV. CASE ORGANIZATION

The case organization designs software for network protocol analyzers. One of the organization's sites in Finland started their agile transformation with pilots during the spring of 2010. They further changed that site's organization of around 30 employees to an agile way of working in the beginning of fall of the same year by starting to follow the methods of Scrum development [1]. During 2011, after initial results and experiences in Finland, similar processes were taken in use at a development unit in India and were planned to be taken in use in other sites as well.

The *Lean and Agile Deployment Survey* was conducted twice in the organization. The first survey took place after the agile methods had been taken in place in Finland and had been in place for about 6 months. The second survey was conducted 12 months later and was expanded to include the site in India, which had adopted similar agile practices during that time.

The targets of the survey assessment were i) to review the current status of agile adoption at two of the case organization's sites, ii) to see how the unit in Finland had been progressing with agile methods between the two survey rounds, iii) to identify focus areas for continuous improvement efforts and iv) to receive feedback on the impressions and assumptions on agile and Scrum processes in other sites of the organization.

To obtain results for the last goal, the survey was also conducted in a third site, which had not yet fully adopted similar processes as the two case sites. The responses of the third site are omitted from the results presented in this publication.

The total number of respondents for the first round in Finland was 25. For the second round, there were 62 responses in total, 25 responses from Finland and 37 from India.

V. RESULTS

The survey was very successful in terms of response rate, which was a full 100 percent in the first round and 80.5 percent in the second round. The high response rate was

 ISBN: 978-1-61208-367-4

attributed to the close collaboration between the case organization and researchers and extensive communication to the survey participants. Personnel of the case organization also sponsored the survey noticeably, so its conduction was well received.

TABLE I. RESPONDENT EXPERIENCE

	How many years of experience in software industry do you have?			
	Round 1 (Finland)	***Round 2 (Finland)***	***Round 2 (India)***	***Round 2 (Total)***
None	0	0	0	0
Less than 2	1	0	6	6
2-5	4	2	13	15
5-10	5	5	16	21
10-20	13	16	2	18
More than 20	2	2	0	2
Total	25	25	37	62

A comparison of the respondents' experience shows that the personnel that participated in the survey were generally very experienced in software development (see Table I). There is also some difference between the experiences between the two sites. Many respondents in Finland had over a decade of experience in software development, which may amount to some opinions reflected in the survey results.

TABLE II. RESPONDENT ROLES

	Round 1 (Finland)	***Round 2 (Finland)***	***Round 2 (India)***	***Round 2 (Total)***
Developer	13	16	18	34
Tester	4	1	10	11
Product owner	4	4	2	6
Scrum master	3	2	6	8
Other	1	2	1	3
Total	25	25	37	62

Most of the responses in the survey came from developers and testers (see Table II). The other roles with significant number of responses were the product owner and Scrum master. As the focus of the survey was at the implementation of agile development process, the responses from these roles also provides a solid basis for the analysis of the results.

Because of the case organization's preference, the statements were evaluated by the respondents on a four-point scale, with an additional option of 'I don't know' instead of a 5-point Likert-type scale [14] usually utilized with the *Lean and Agile Deployment Survey*. The answering options with corresponding weights used in average calculation in the following results section were as follows (see Table III).

TABLE III. SURVEY ANSWERING OPTIONS

Option	**Option weight**
Disagree	1
Partially agree	2
Largely Agree	3
Fully Agree	4
I don't know	-

The following tables and figures present selected findings from the survey which may be interesting in the context of multi-site agile adoption. The results for individual statements (see Figures 1-10) are presented as the distribution of responses and an average result in the statements in four separate rows. The first row presents the results received in the first survey that was conducted around 6 months after the agile adoption had taken place in the Finland unit. The second and third row include responses 12 months later from the Finnish and Indian units, respectively. The final row shows the combined answers in the second survey round from both sites (Finland and India).

Please note that the 'I don't know' –answers are not included in the average calculations. However, in some statements the amount of 'I don't know' –responses itself is significant.

Firstly, a very interesting finding can be made by looking at the collective average of the overall responses between the two survey rounds (see Table IV).

TABLE IV. SURVEY AVERAGE

	Round 1 (Finland)	***Round 2 (Finland)***	***Round 2 (India)***	***Round 2 (Total)***
Response average	3,04	2,76	3,28	3,07

The fact that the average score in Finland in the second survey is lower than 12 months earlier is an alerting sign, as the statements were formed in a positive form in accordance of the case organization's process description. There was some indication from the case organization that they had not had sufficient resources to actively react to issues raised in the first survey and subsequent retrospectives during the 12 month period between the two surveys. A possible cause for the reduction in the average results may also be increased experience and awareness in the agile methods. This could affect the results as people become more aware of their processes and the issues concerning them than before.

Also, perhaps surprisingly, the average score in India was much higher than it was in Finland as seen from the second round average scores. Several reasons may affect this difference, with cultural reasons perhaps being the most obvious explanation.

 ISBN: 978-1-61208-367-4

Reasons for the drop in score are evident in some survey results. One main improvement area for the case organization based on the first survey was the lack of identified value of continuous improvement activities (see Figures 1 and 2).

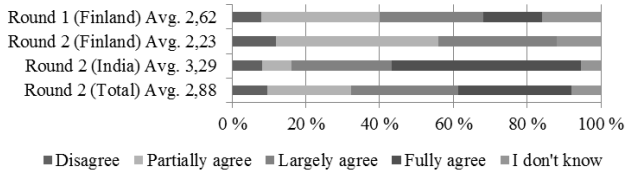


Figure 1. Scrum teams change their ways of working based on retrospectives.

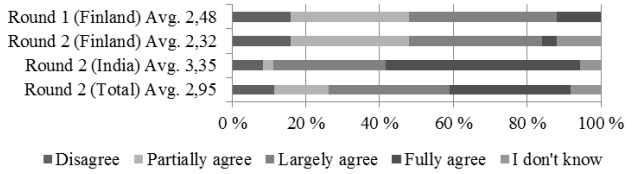


Figure 2. We reduce wasteful activities frequently.

The lack of resources assigned for following up on this improvement area show as reduced results in the second round in the Finland unit. Again, results on the topic are higher in the India site.

A second major improvement area identified based on the first survey round was the lack of measured and communicated evidence of the benefits of the agile methods for the organization (see Figures 3-6).

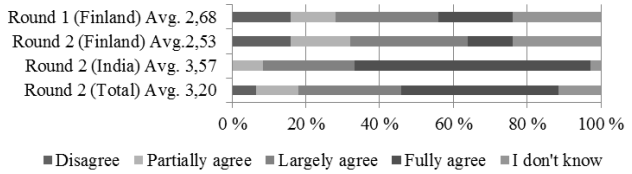


Figure 3. I am more productive with the agile way of working.

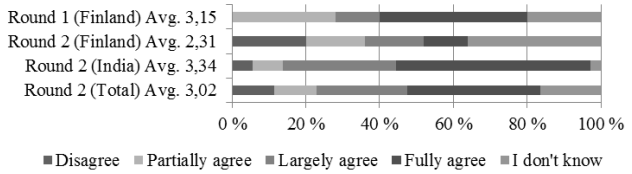


Figure 4. We are more productive as a Scrum team.

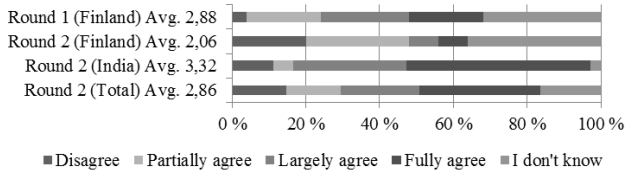


Figure 5. Product quality has been improved by applying agile development.

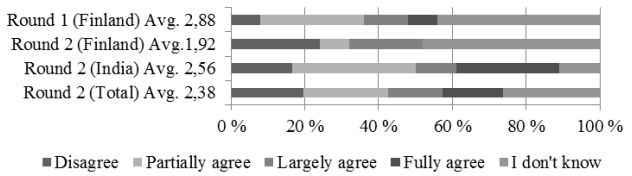


Figure 6. Development time has decreased by applying agile development.

An action point after the first survey round was to provide the teams more information on the benefits of agile in comparison with earlier working methods. This issue had apparently not received enough attention because the second survey round indicated some decrease in results on the matter as well as an increase in 'I don't know' –responses in Finland. Another possibility for the results is that the quality and productivity have actually not been improved with the new methods. The measuring of the benefits of agile is a very interesting and difficult topic among all organizations implementing the methods, but high consideration should be used on how to provide teams more information on actual benefits of agile.

There was also possible need for further training within the organization (see Figures 7 and 8).

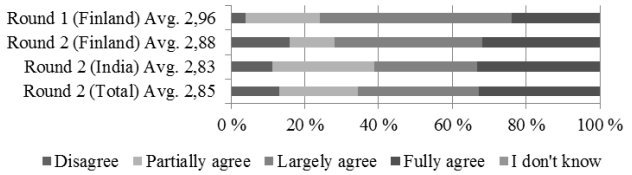


Figure 7. I have received enough training for carrying out my work.

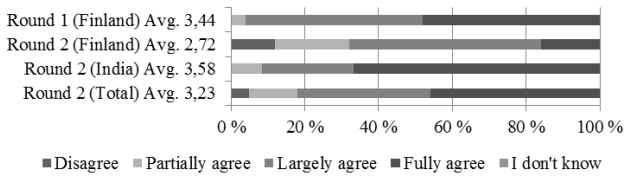


Figure 8. I feel confident with myself with the agile way of working.

When comparing the results between the sites in Finland and India, it can be seen that the training needs appear to be equally divided between the two sites. However, there is a noticeable difference between the sites in the confidence in individual capabilities. This can possibly again be explained by cultural differences.

There was also some difference in statements about the preference of team co-location between the sites. There is a noticeable change in the answers between Finland and India (see Figures 9 and 10).

 ISBN: 978-1-61208-367-4

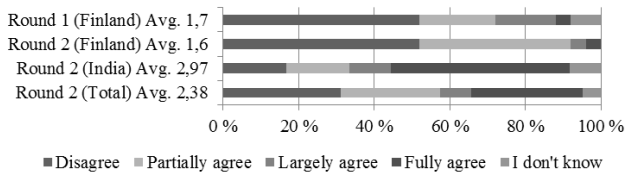


Figure 9. I prefer to work in a multisite Scrum team.

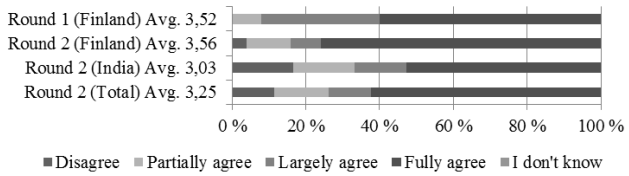


Figure 10. I prefer to work in a local Scrum team.

Differing from the answers in India, there seems to be a clear preference to co-location of team members in Finland. The co-location in generally viewed as an important part in Scrum processes and the results in Finland show the preference that has come by experience in that site. The conflictingly high results of India in both of the two tables above may involve cultural influences, but also some lack of experience since the agile methods had been in use there for a shorter period of time.

An additional interesting comparison was made between the two survey rounds in the overall amount of 'I don't know' –answers (see Table V).

TABLE V. PERCENTAGE OF 'I DON'T KNOW' RESPONSES IN ALL STATEMENTS

Round 1 (Finland)	*Round 2 (Finland)*	*Round 2 (India)*	*Round 2 (Total)*
9,8%	12,7%	7,62 %	9,8%

In the second survey round, the amount of 'I don't know' -answers in Finland is quite a lot higher than in India. When results between the two rounds are compared, we find that the percentage in Finland has increased between the two rounds and that the percentage in India is even lower than Finland in the first round. There was a similar amount of time elapsed from the agile adoption in Finland in the first round and India in the second. This could indicate that the amount of knowledge acquired during the 12 months between survey rounds in Finland lead to an increase in awareness of issues, or to some other reasons which lead to this result.

VI. DISCUSSION

Based on the survey results, the main improvement areas identified in the first survey round were not given enough attention after conducting the survey. This was also admitted by the case organization because of reduced resources for the improvement efforts. This is one of the main reasons why the results in the Finland site appear lower in the second round.

However, the other main reason for the reduction in response averages in some statements is believed to be increased awareness on the topic of agile methods and possible issues related to them. The combined average result in all statements between Finland in round 1 and India in round 2 are similar. The amount of time that these two sites had been using the agile development methods before their first respective survey rounds was also similar.

The first important improvement suggestion for the case organization in the opinion of the researchers was to improve the resources currently utilized for change management and improvement efforts. The teams may need more support and resources for successful organizational transformation. This should include more support for continuous improvement activities and the follow up of these activities, since there were no definitive improvements that could be identified from the first assessment round.

The identified decrease in results should be taken seriously to see what kind of improvement actions could be taken. This should also include very active participation from all members of the development organization, since they will be most aware of the issues regarding their daily work. The practices and processes that do not work should be adapted according to the organization- or unit-specific preferences while remembering to include the agile principles and mindset.

Continuous improvement activities should have a strict process to follow, which includes communication to all interested stakeholders on the progress of the activities and a responsible individuals who have allocated time to conduct the activity. Many additional success factors can support the sustainability of improvement activities as well, which should be kept in mind when implementing changes [15].

The follow-up of the activities should also include a larger scale follow-up of the adoption of agile methods. Some forms of quantitative or qualitative measurements of the possible benefits of agile (in productivity, quality, etc.) should be measured and communicated in all units, including the sites that may take the agile methods into use in the future. This shows that the organization is committed to the changes and that the activities that are requested of the members of the organization have justifications behind them. There was already some evidence of doubt in the agile methods in the first survey round and these doubts should be addressed properly through discussion.

In addition to the assessment results changing with time elapsed between the organizational change and the assessment, the results of the survey also indicate bias in the results based on cultural differences. When assessing the success of multisite organizational changes, it should be noted that the results may vary between locations for reasons that may not be possible to affect with any change management processes. Therefore, it may be useful in some

 ISBN: 978-1-61208-367-4

cases to assess different global sites individually, instead of comparing the results of sites between each other.

VII. CONCLUSION

The results of this research can be used by researchers and practitioners when assessing organizational changes. Assessment results between geographically distributed sites may not always be directly comparable between each other. Cultural differences in results and the difference in elapsed time from the organizational change may also affect assessment results and should be noted when analyzing data.

It would also be beneficial to compare results of a similar assessment with a different scaling method, like e.g., the Likert-type scale. The scaling itself should not be a contributing factor in this study, but additional assessment cases with similar backgrounds could be used to validate the influence of the used survey scale.

The assessment process could be repeated in the case organization for a third time to analyze further progress of the organizational change. The findings of this assessment were used to focus future improvement efforts in the case organization and to provide feedback on how they understand their agile transformation so far. The results were presented to all participants through an open discussion session by the researchers and a written report was communicated openly inside the organization. The report was also brought into general knowledge by giving access to it within the organization.

ACKNOWLEDGMENT

The authors would like to acknowledge the contribution of the Finnish Cloud Software Program [13] for the funding received for this research.

REFERENCES

[1] K. Schwaber and M. Beedle, "Agile Software Development with SCRUM", Prentice Hall, 2001.

[2] K. Beck, "Embracing change with extreme programming" Computer, Volume 32, Issue 10, pp.70-77, 1999.

[3] K. Beck, et al., "Principles behind the agile manifesto" [Online] Available from: http://agilemanifesto.org/principles.html 2014.08.12.

[4] C. Larman and B. Vodde, "Scaling Lean & Agile development. Thinking and Organizational tools for Large-scale Scrum. " Addison-Wesley, USA. 2009.

[5] D. Leffingwell, "Scaling software agility: Best practices for large enterprises." Pearson Education, USA, 2007.

[6] J.D. Herbsleb and D. Moitra, "Global software development". IEEE Software, Volume 12, Issue 2, pp.16-20, 2001.

[7] D. Anderson, "Kanban - Successful Evolutionary Change for your Technology Business", Blue Hole Press, USA, 2010.

[8] J. Bosch and P. Bosch-Sijtsema, "From integration to composition: On the impact of software product lines, global development and ecosystems", The Journal of Systems and Software, Volume 83, Issue 1, pp. 67-76, 2010.

[9] P.J. Ågerfalk, et al., "A framework for considering opportunities and threats in distributed software development.", Proceedings of the International Workshop on Distributed Software Development, Paris, France. Computer Society, 2005. pp. 47-61

[10] A. S. Alqahtani, J. D. Moore, D. Harrison, and B. Wood, "Distributed agile software development challenges and mitigation techniques: A case study." The Eight International Conference on Sofware Engineering Advances, (ICSEA 2013) IARIA pp.352-358, ISBN: 978-1-61208-304-9.

[11] T. Oliveira and M. Silva, "Method for CMMI-DEV Implementation in Distributed Teams" The Sixth International Conference on Software Engineering Advances (ICSEA 2011) IARIA pp.312-317, ISBN:978-1-61208-165-6.

[12] S. Misra and L. Fernández-Sanz, "Quality Issues in Global Software Development", (ICSEA 2011) IARIA pp.325-330, ISBN: 978-1-61208-165-6.

[13] Cloud Software Finland. [Online]. Available from: http://www.cloudsoftwareprogram.org/cloud-software-program 2014.08.12.

[14] R. Likert. "A Technique for the Measurement of Attitudes". Archives of Psychology Vol. 140, pp. 1–55, 1932.

[15] N. Nikitina and M. Kajko-Mattson, "Success factors leading to the sustainability of software process improvement efforts." The Sixth International Conference on Software Engineering Advances (ICSEA 2011) IARIA pp. 581-588, ISBN: 978-1-61208-165-6.

 ISBN: 978-1-61208-367-4

Towards Agile Composition of Service Oriented Product Lines: A Mashup-based Approach

Ikram Dehmouch, Bouchra El Asri, Zineb Mcharfi
IMS Team, SIME Laboratory
ENSIAS, Mohammed V Rabat University
Rabat, Morocco
{ikram.dehmouch@gmail.com, elasri_b@yahoo.fr, zineb.mcharfi@gmail.com}

***Abstract*—Large scale product lines cover multiple domains with different concepts and concerns. Thus, involving domain users in the development life cycle is a key factor for the success of the composition process combining the different subdomains of the intended resulting large scale system. In fact, domain users master the domain concepts, the scope of each subdomain and the interactions between the different subdomains to be composed. This makes them key actors in the composition process. Adopting agile principles is then required to offer intuitive and simple composition techniques for end-users. One of these emergent techniques is Mashup, which is mainly concerned with web service composition in an ad hoc way. This paper proposes using a Mashup component as an underlying composition technique for large scale service oriented product lines in order to bring agility to the process of composing the different subdomains services. The proposed Mashup component in allows incremental composition to achieve agility and to address scalability issue as well.**

Keywords—Product Line Engineering; Feature Model; Agile Software Development; Service Oriented Computing; Mashup

I. INTRODUCTION

The Product line approach has been already successfully applied to various industrial domains, such as avionics [1] and automotive systems [2], etc. With regard to software engineering, Software Product Line Engineering (SPLE) constitutes a major advance, as it allows building software from a set of previously developed and tested parts, based on the domain knowledge. This generates considerable benefits in terms of time, quality and resources [3].

However, traditional SPLE is no more enough to face modern applications, which tend to be cross-industry and to cover multiple domains simultaneously. This has led to the advent of the large scale product lines concept combining various subdomains with heterogeneous crosscutting concerns such as health, telecommunication, transport, etc. Thus, composing these subdomains to generate the intended large scale system is becoming a crucial concern [4]. The composition process in such systems becomes more problematic, since it should scale to big development projects sizes with hundreds of users/developers from several fields.

On the other hand, domain users are an important ingredient of this composition process success, since they master the scope and the different concepts within the subdomains to compose. Thus, one possible way to address the scalability issue is to involve end-users in the composition process.

Consequently, opting for Agile Software Development (ASD) in combination with SPLE is the key to make the composition process simple and intuitive for end-users, solve possible conflicts that may occur in such heterogeneous crosscutting environments and allows incremental development, which meets the scalability issue.

In fact, ASD put end-users at the heart of the software development process, since it is based on constant interaction with customers [5].

In this context, we propose in this article an agile composition approach for large scale product lines based on a consumer-centric technique called "Mashup". In fact, Mashup is an extremely consumer-centric and lightweight service composition technology [6], which can be exploited to address scalability issue throughout bringing agility to the proposed product line composition approach.

To explore how Mashup facilitates this service composition, this paper is organized as follows: In Section 2, we present some basic concepts related to two main paradigms involved in our work, which are SPLE and ASD. Section 3 draws up the motivations of our work. Section 4 discusses about the related work. Section 5 presents an overview of our approach. A motivating scenario showing the interest of our approach is described in Section 6. Finally, Section 7 concludes the paper.

II. BASIC CONCEPTS

- *SPLE*

SPLE is an emergent paradigm which is the result of bringing the reuse-based product line concept, adopted mainly in industry, to software engineering development process [7]. It consists of constructing software products from reusable core assets. This results in lower costs, shorter time-to-market, and higher quality, since a family of products is generated instead of developing them one by one from scratch [8].

According to Kang et al. [9], the product line is defined as a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission, but that still show distinct and different characteristics.

The SPLE process is usually divided in two main complementary phases: *Domain engineering* and *Application engineering* [3]. While the first one deals with the development and maintenance of reusable core or domain assets, the second one is about using those assets in order to build individual software products. The first step in domain engineering is *business scoping* which is performed using specific models called Feature Models (FM). The notion of FM was proposed by Kang et al. [9] to represent commonalities and variabilities among the products within the same domain. In fact, FM is a tree structure representation of features—*"a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems*—[9] and the relationships between them. A feature can be either mandatory, optional, alternative (Xor group) or part of an Or group. Thus, multiple products can be built form a set of reusable assets depending on which alternative was selected

 ISBN: 978-1-61208-367-4

during product configuration. This leads us to a core principle in SPLE, which is Variability [3].

Domain engineering includes also the definition of a *reference architecture*, and the development of *Software reusable components*. At application engineering level, the FMs, reference architecture and software reusable components are then used as a basis for deriving a specific domain application to meet needs of a specific end-user.

Thus, profitable SPLE requires an extensive up-front investment to develop reusable domain assets for later efficient use in products. This carries the risk of not getting a viable return on investment if the pre-developed assets are not sufficiently reused [10]; hence the need to resort to ASD.

- *ASD*

Most of today's systems are evolving towards community-driven development approaches where the end-users are involved in the whole development life cycle. This kind of software approach is known as ASD.

According to Larman [11], ASD is based on short iterations. Each one is a self-contained, mini-project with activities that span requirements analysis, design, implementation, and test. This allows taking into account feedback from users in iteration N so that needed refinements and adaptations are made in iteration N+1. Hence, ASD give mush importance to people and put them at the heart of the development process.

Besides, research has shown that shorter iterations have lower complexity and risk as they are concerned with small fragments of the system, and allow then better feedback, higher productivity and higher success rates [9]. One other major characteristic of ASD is that it is basically built on response to change rather than change prevention [5], which fits the changing nature of software development in which requirements, technology and development team are in constant change.

ASD is based on four fundamental values and twelve principles as presented in The Manifesto for Agile Software Development written by a group of software consultants in 2001 [5]. Most of agile methods such as Extreme Programming (XP) and Scrum [12] share these principles, which are basically about frequent communication, frequent deliveries of working software increments, short iterations and active customer engagement throughout the whole development life-cycle.

III. MOTIVATIONS

The wide scope covered by large scale Software Product Lines (SPL) makes their management a very complex and tedious task. One efficient way of making this task easier and better mastered is the decomposition of these large scale systems in smaller subdomains, each one covering a specific field and involving only business users concerned with this field. To get a final product variant of the large scale product line, the corresponding feature models of the different subdomains should be composed.

This composition process has several limitations when it comes to cross-domain large scale systems. One major limitation is that it would not be obvious to adopt a traditional SPLE approach based on up-front development to assure valid compositions. Thus, scalable product line composition represents a central motivation for our work.

We believe that adopting ASD in this composition process is the key to address the scalability issue. Three arguments motivate our choice:

- ASD allows incremental composition: this meets the modularity logic consisting of decomposing the large scale system into many subdomains and then recomposing them in an incremental way to get a specific product variant.

- ASD is consumer-centric: this allows better management of the different stakeholders regardless of their heterogeneity. Besides, users are involved in the whole development life cycle which reduces the extensive up-front investment.

- ASD and SPLE combination generates many benefits: On one hand, some of the central agile practices may increase flexibility and customer collaboration. On the other hand, the concepts of SPLE are needed in order to manage the diversity of products, the large customer base, and the long-term perspective, which are the characteristics of managing and developing a product line over time.

IV. RELATED WORK

At first sight, ASD and SPLE seem to be contradictory approaches, since SPLE is a proactive approach which requires planning the development of assets in advance for later reuse, in contrast to ASD, which is a reactive approach that avoids up-front planning and development throughout perpetual interactions with end-users.

Though, several experimentations showed that there is a great interest in combining SPLE and ASD approaches [13]. Two case studies driven by Ghanam and Maurer [14][15] show that, besides being practically feasible, the combination of some XP practices and SPLE reduces rework and the cost of producing customized solutions, since it enables customers involvement.

Composite Feature Models (CFM) is another concept combining SPLE and ASD. According to Urliand et al. [16], CFM are an extension to classic Feature models, since these latter are not powerful enough to handle agility challenges. *Separation of concerns* is one of the main pillar on which CFM are built. It offers end-users simple views on the system, since they focus only on their domain concepts without being overwhelmed by the other domain concepts. CFM concept is also based on *bottom-up modeling*. In fact, users have the possibility to change their requirements at any point of the development life cycle. This modification is then introduced in the corresponding partial feature model and an automated algorithm is used to merge the modified partial FM into the CFM [17]. Finally, *automated refactoring* allows CFM handling vocabulary mismatch due to the heterogeneity in face to face conversations with different groups of users, which is an important agility issue.

On the other hand, some other researches show a growing interest in how Service-Oriented Computing (SOC) [24] can be adopted as a mean for enhancing agility and flexibility in SPLE. Kotonya et al. [18] propose a consumer-centered approach combining SPLE and SOC through the two following main steps which are *Feature analysis* for representing different services involved into a family of Business Processes, and *Service analysis* in which dynamic services are selected depending on whether the corresponding features are selected or not in the FM configuration relevant to a specific Business Process (BP). Cubo et al. [19] have also developed DAMASCo framework (model-based service-oriented architecture approach that makes the design,

 ISBN: 978-1-61208-367-4

development and deployment of processes more agile) in combination with feature models to safely handle the variability in the service composition at runtime. Thus, if the client request changes, a new valid configuration of the product family containing the required features is automatically created.

Dynamic adaptation is another advantage of combining SPLE and SOC. In fact, Alférez et al. [20] propose a framework that uses variability models to support the dynamic adaptation of service composition. These variability models describe the dynamic configurations of the service composition in terms of activation or deactivation of features. The information captured in these models is combined with context model, which collects context knowledge, and composition model describing the service composition. This combination is performed through the weaving model, which connects the variability model to the composition model based on the context model.

Some other works deal more with the scalability issue in SPLE regardless of the agility aspect such as Dhungana et al.'s work [21], which is about the System of Systems (SoS) paradigm [4] (i.e., systems designed and constructed by combining several heterogeneous subsystems that are themselves composed of many components, data structures and service, etc.) The composition process proposed in this approach is performed through two injection mechanisms *push* and *pull* that allow generating in a flexible way a conjoint model representing a common model of the selected components in the SoS, which can be deployed in a target platform.

Table 1 shows a comparative assessment of works above based on our motivations:

TABLE I. COMPARATIVE ASSESSMENT

Related work	Handling scalability	Handling agility	Adaptability to Service Oriented product line	Dynamic adaptation
[14][15]	-	++	-	-
[16][17]	+	++	-	+
[18][19][20]	-	++	+++	+
[21][4]	+++	-	-	-

V. AN AGILE MASHUP-BASED COMPOSITION APPROACH FOR LARGE SCALE PRODUCT LINES

A. Approach overview

To address the scalability issue in the SPL composition process while taking into account the ASD principles, our approach uses a mix of both SPLE and SOC paradigms. On one hand, SPLE brings a valuable knowledge about variability within the large scale product line throughout the whole development lifecycle. Consequently, late variability is also handled allowing users to specify the services to compose even at runtime. On the other hand, SOC allows loose coupling among interacting services, which enables flexible and agile service composition. Besides, its dynamic nature can be exploited to guide dynamic adaptations at runtime in order to fulfill specific business objectives according to context information especially in terms of availability of dynamic services, i.e., services that can be invoked only at runtime (e.g., real-time information services about current weather).

As our approach is an SPL-based one, it should cover the two main phases of SPLE, which are domain engineering and application engineering.

From a domain engineering view, our approach proposes to use a specific FM notation to distinguish dynamic services from static ones in order to define the scope of dynamic adaptations, which can take place at runtime. To this, features corresponding to dynamic services are represented within a dotted box in the FM. One other major information that should be also captured at this level of SPLE is the standard business workflow of service orchestration. We propose to represent this information using a BPMN 2.0 model [22], as it is a widely used standard for BP definition, not only as a graphical representation, but also as an execution language. Besides, it is a user-friendly model, which consolidates best practices from different modeling techniques such as UML Activity Diagram, IDEF, ebXML BPSS, Activity-Decision Flow (ADF) Diagram, etc. As covering all possible cases in this generic workflow is a very tedious task, we propose to use FM in order to represent all the variation points of the BPMN 2.0 model using alternative features. Besides, domain engineering includes also the definition of a service oriented reference architecture and the development of reusable BPEL fragments corresponding to the reusable parts of BP.

From an application engineering view, as our contribution is a user-centered approach, it is the end-user who defines the desired product configuration based on his needs. Thus, the generic BPMN 2.0 model is refined according to this specific need, the variation points are resolved, and the corresponding BPEL code and the applicative architecture are generated. In the following section we give more details about how our approach brings agility to the composition process of service oriented product lines.

B. Agility and Mashup

Agility is the central added value of our contribution. To fulfill this, we propose using Mashup as an underlying service composition technique, since it is a lightweight and quick way to integrate multiple sources of applications into a single one, supporting programming for end-consumers without complex environment. In fact, we can take advantage from the Mashup component proposed by Liu et al. [6]. It is composed of three main parts, which are: User Interface (UI) component, Service component and Action component. Adopting this Mashup component allows bringing more agility to our proposed approach through three main principles, which are: Separation of concerns, Dynamic adaptation and Incremental development. Hereafter, we develop each one of these principles:

- Separation of concerns

As the large scale product line is a cross domain system, we propose to decompose it into several subdomains. To emphasize the agility principle, our approach involves end-users from the earlier steps of the development lifecycle. As depicted in Figure 1, each subdomain is represented using a swimlane and it is managed by a group of domain users and experts, as they are the best placed for defining domain feature models configurations and BP instances, as explained in the previous section. It is the UI component of the Mahsup component who offers the end-users a user interface to perform all those definitions and to transmit this information about the services retained and the workflow orchestrating them to the Action component.

 ISBN: 978-1-61208-367-4

Besides being an agile principle, separation of concerns allows also better control of large scale systems, which meets our scalability issue.

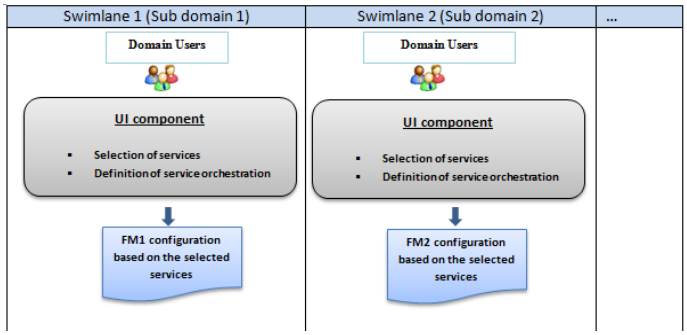


Figure 1. Separation of concerns.

- Dynamic adaptation

Distinguishing dynamic services from static ones in FM offers our approach the possibility of dynamic adaptation at runtime according to context changes. In fact, in contrast to static services, getting the accurate information about the availability of dynamic services providers can only be performed at runtime. Based on this information, the final variant of the sub-product line (subdomain) is generated.

But, there are several previous steps before achieving this generation:

a. Generation of an FXML file from the FM configuration: Based on the information provided by the UI component about the FM configuration (service selection), an XML file is generated corresponding to the current user's requirements called FXML. In fact, we propose that each feature in the FM configuration has its corresponding XML element. To distinguish dynamic services from static ones in the FXML, we propose to use two kinds of xml tags: <dynamic_service> for dotted boxes in FM configuration and <static_service> for solid ones.

b. Parsing FXML file: At this step, the Action component takes as input the generated FXML file and the BPEL 2.0 code corresponding to the generic BPMN 2.0 subdomain model (developed at implementation phase of domain engineering level). In fact, the Action component parses the FXML file based on a specific mapping relating FXML elements to <invoke> BPEL elements. If a service has its XML element retained in FXML file, then its corresponding invoke BPEL fragment is kept in the final BPEL file, else it is removed. Besides, certain fragments of the generic BPEL might be moved to respect the order required by the end-user. Thus, the Action component defines three actions: *add*, *remove* or *move*, which are used in order to invoke the right services in the right order according to the user's needs. The action component eliminates the variation from the final BPEL. In fact, each variation point is represented by a variable in the generic BPEL. Once the needed service is selected in the FM configuration, the variable is set to the selected value.

c. Checking service provider's availability: Before generating the final BPEL file, the Action component sends a request to the Service component in order to check the service provider availability at runtime. Thus, if the service provider is available then the corresponding BPEL fragment is kept in the final BPEL file else it is removed. Thanks to this, context changes are handled by our approach allowing dynamic adaptations at runtime.

d. Generation of the new variant of the sub-product line: Once the final BPEL file is constructed, it is executed by the service component in a specific execution engine and the result is sent to the Action component. This latter updates the user interface by returning the result to the UI component.

Figure 2 presents the details about the proposed Mashup component and the different steps covered before the generation of the sub-product line variant:

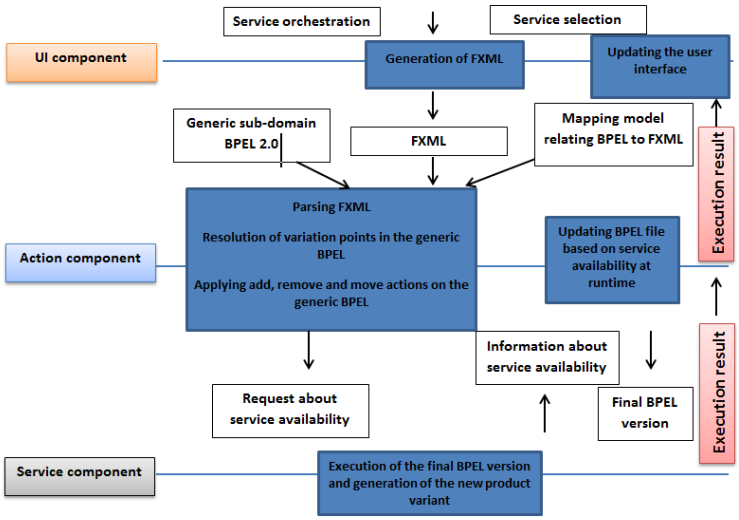


Figure 2. Intra-domain Mashup component.

- Incremental development

At this stage, we have a product variant at the output of each swimlane. This output is validated by the domain users, since it has been produced according to their definition of BP. We propose that each one of these output is composed with the following one in an incremental composition process until covering all swimlanes, and thus, covering the large scale product line, as depicted in Figure 3.

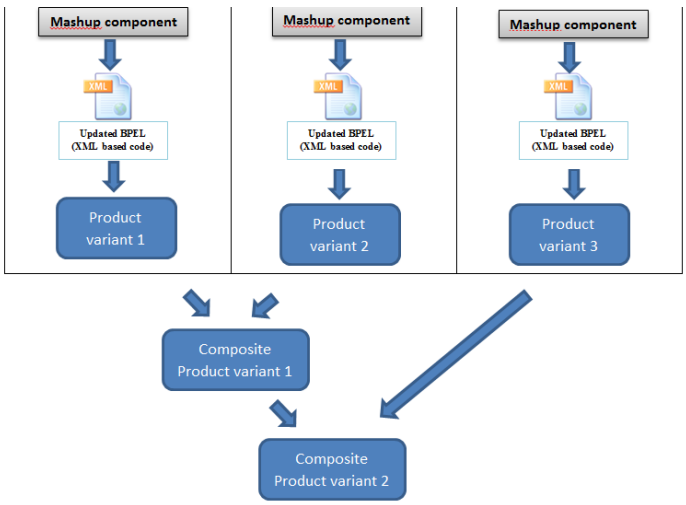


Figure 3. Inter-domain Mashup-based composition.

 ISBN: 978-1-61208-367-4

The underlying composition approach in this inter-domain composition phase is the same Mashup component. However, there is a slight difference, which is the adoption of design by contracts [23] as a set of pre- and post-conditions annotating the BPELs 2.0, relevant to the sub-product lines to be composed, at the input of the inter-domain Mashup component. In fact, design by contracts allows defining the interconnection order rules between the different subdomains, as their respective domain users are not intended to know these cross domain rules. The Action component uses these latter to apply the action *move* in order to put the services in the right order in the output composite BPEL 2.0. In fact, if all pre-conditions of a BPEL 2.0 fragment relevant to the first subdomain are fulfilled, the appropriate BPEL 2.0 fragment, relevant to the second subdomain, is invoked and placed at a specific binding point in the resulting composite BPEL 2.0.

VI. MOTIVATING SCENARIO: DIABETES SELF-MANAGEMENT SYSTEM

We choose as a motivating scenario to demonstrate the results of our approach the Diabetes self-management system, i.e., a system allowing diabetes patients to do a regular monitoring of their health and of the different risk factors, which may influence their disease and imply complications.

In the following, we present the three main steps of applying our proposed approach to this example. These steps are:

Separation of concerns: the first step consists of decomposing the Diabetes self-management system in two main subdomains, which are telecommunication domain and health domain managed by patients and doctors respectively. At this level, the UI component offers doctors a specific user interface in order to define the objectives that should be fulfilled throughout the daily treatment and monitoring proposed to their patients based on their criticality degree. These objectives are represented as features in the health subdomain FM such as taking insulin injections or tablets, performing health records (e.g., blood pressure, blood glucose, etc.), walking during thirty minutes, note unusual symptoms, etc. The doctor's feature selection is then transmitted to the Action component in order to determine which services to invoke and in which order. In fact, once the treatment objectives are selected, their corresponding order can be retrieved from the generic BPMN 2.0 model as it has already been defined by the doctors.

On the other hand, another user interface allows patients to choose the most suited way of interaction with their doctors, (e.g., Telephone, SMS, Interactive Voice Responder (IVR), etc.), as a means of telecommunication. Besides, service orchestration is also possible for patients throughout the definition of the interaction frequency with their doctors via the specified mean of telecommunication. For example, if a patient chooses IVR as mean of interaction, he should define a specific schedule of virtual home visits accomplished via IVR based on his availability.

Dynamic adaptation: at this level, FXML files corresponding to both health and telecommunication subdomains are generated based on the information provided by the UI component. The action component then parses the FXML files in order to update the generic BPELs files. For example: for a specific patient, the daily treatment consists of making a blood glucose record, sending the record result to the doctor in charge, receiving response and applying doctor's recommendations (i.e., taking a tablet of a specific medicine, walking during twenty minutes, visiting the doctor, etc.), etc. The generic health BPEL is then updated according to this order.

In order to send the appropriate recommendations to the patient, doctors need the accurate values of health records. These values could be transferred instantly to the system via mobile recording devices such as Personal Digital Assistant (PDA). According to our approach, the next step consists of checking the availability of the service, which collects the health records information from the mobile device at runtime in order to generate the right health BPEL.

Incremental development: According to our approach, we have as a result two BPELs, each one corresponding to one subdomain. To generate the composite diabetes self-management variant, we use the pre- and post-conditions annotating the input BPEL fragment at the input of the intra-domain Mashup component. For example, the BPEL fragment corresponding to the IVR services cannot be invoked until the patient notices an unusual symptom; else the system simply sends SMS to remind the patient about medicines and regular health records.

VII. CONCLUSION AND FUTURE WORK

Combining ASD and SPLE has proven to be a worth exploring track as it generates many advantages in terms of reduced time to market and valuable return on investment.

In this paper, we proposed to take advantage from this combination in large scale product lines composition. The main finding was that bringing agility to the composition process throughout the Mashup component ensures the scalability of our composition approach. In fact, the iterative and incremental nature of ASD allows modularity and thus a better control of each sub-system of the large scale system. On the other hand, the user-centric nature of ASD involves only domain users concerned with the appropriate subsystem, which optimizes the time and cost of development.

As users are put at the heart of our agile approach, the main challenge of our future work is dealing with the perpetual changes reflecting the new user requirements. Our future work will then emphasize on the definition of a weaving model relating the FM to the BPMN 2.0 model, in our proposed Mashup component, in order to ensure the repercussion of the new user requirements on the resulting composite product variant.

REFERENCES

[1] F. Dordowsky, R. Bridges and H. Tschope, "Implementing a software product line for a complex avionics system". In 15th International SoftwareProduct Line Conference (SPLC), IEEE, 2011, pp. 241-250.

[2] S. Thiel, S. Ferber, T. Fischer, A. Hein, and M. Schlick, "A case study in applying a product line approach for car periphery supervision systems", (No. 2001-01-0025), SAE Technical Paper, 2001.

[3] K. Pohl, G. Böckle, and F. Van Der Linden, "Software product line engineering". Springer, 10, 2005, pp. 3-540.

[4] M. Jamshidi, "Systems of Systems Engineering: Principles and Applications".Taylor & Francis, ISBN 9781420065893, 2010.

[5] M. Fowler and J. Highsmith,"The agile manifesto. Software Development",9(8), 2001, pp. 28-35.

[6] X. Liu, Y. Hui, W. Sun, and H. Liang, "Towards Service Composition Based on Mashup," 2007 IEEE Congr. Serv. (Services 2007), Jul. 2007, pp. 332–333.

 ISBN: 978-1-61208-367-4

[7] F. J. Linden, K. Schmid, and E. Rommes. “Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering”. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[8] D. Benavides, S. Segura, and A. Ruiz-Cortés, “Automated analysis of feature models 20 years later: A literature review”, Information Systems, 35(6), 615-636, 2010.

[9] K. C Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, “Feature-oriented domain analysis (FODA) feasibility study” (No. cmu/sei-90-tr-21). Carnegie-Mellon Univ Pittsburgh, Software Engineering Inst, 1990.

[10] I.S. Jacobs and C.P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G.T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271-350.

[11] C. Larman, “Agile and Iterative Development: A Manager's Guide”. Boston: Addison Wesley, 2004.

[12] R. C. Martin, “Agile software development: principles, patterns, and practices”. Prentice Hall PTR, 2003.

[13] G. K. Hanssen and T. E. Fægri, “Process fusion: An industrial case study on agile software product line engineering”. Journal of Systems and Software, 81(6), 2008, pp. 843-854.

[14] Y. Ghanam and F. Maurer, “Extreme product line engineering: Managing variability and traceability via executable specifications” In Agile Conference, 2009. AGILE'09, August. 2009, pp. 41-48.

[15] Y. Ghanam and F. Maurer, “Extreme Product Line Engineering–Refactoring for Variability: A Test-Driven Approach”. In Agile Processes in Software Engineering and Extreme Programming , Springer Berlin Heidelberg, 2010, pp. 43-57.

[16] S. Urli, M. Blay-Fornarino, P. Collet, and S. Mosser, “Using composite feature models to support agile software product line evolution”. InProceedings of the 6th International Workshop on Models and Evolution, ACM, October.2012, pp. 21-26.

[17] M. Acher, P. Collet, P. Lahire, and R. B. France, “Separation of concerns in feature modeling: support and applications”. In Proceedings of the 11th annual international conference on Aspect-oriented Software Development, ACM, March. 2012, pp. 1-12.

[18] G. Kotonya, J. Lee, and D. Robinson, “A consumer-centred approach for service-oriented product line development,” 2009 Jt. Work. IEEE/IFIP Conf. Softw. Archit. Eur. Conf. Softw. Archit., September. 2009, pp. 211–220.

[19] J. Cubo, N. Gamez, L. Fuentes, and E. Pimentel, “Composition and Self-Adaptation of Service-Based Systems with Feature Models”, In Safe and Secure Software Reuse, Springer Berlin Heidelberg, 2013, pp. 326-342.

[20] G. H. Alférez, V. Pelechano, R. Mazo, C. Salinesi, and D. Diaz, “Dynamic adaptation of service compositions with variability models”, J. Syst. Softw., vol. 91, May. 2014, pp. 24–47.

[21] D. Dhungana, A. Falkner, and Haselböck, “Generation of conjoint domain models for system-of-systems”. In Proceedings of the 12th international conference on Generative programming: concepts & experiences, ACM, October. 2013, pp. 159-168.

[22] T. Allweyer, “BPMN 2.0: introduction to the standard for busines process modeling”. BoD–Books on Demand, 2010.

[23] T. Thüm, I. Schaefer, M. Kuhlemann, S. Apel, and G. Saake, “Applying Design by Contract to Feature-Oriented Programming”. In Proceedings of the International Conference on Fundamental Approaches to Software Engineering (FASE), Springer, 2012, pp 255–269.

[24] M. N. Huhns and M. P. Singh, “Service-oriented computing: Key concepts and principles”. Internet Computing, IEEE, 9(1), 2005, pp. 75-81.

 ISBN: 978-1-61208-367-4

An Approach and a Tool for Systematic Review Research

Manuel Gonçalves da Silva Neto, Walquiria Castelo Branco Lins, Eric B. Perazzo Mariz
Recife Center for Advanced Studies and Systems (CESAR)
Recife – PE, Brazil
Emails: {manuel.pi, wcbl, eric.perazzo}@cesar.edu.br

***Abstract*— Systematic Reviews and Systematic Mappings are widely used in medicine in an area called *evidence-based studies*. Recently, these techniques have been adapted and used in secondary studies in the area of Software Engineering and Systems. Sorting and synthesizing information in a particular research area by analysis of their primary studies, both involve both extensive work and researcher dedication. Adapting techniques applied in *evidence-based studies* in the medical field to software engineering led to an approach, which divides Systematic Review tasks into three main phases, namely, planning, conduct of review itself and reporting the results. Unlike in the area of medicine, in which there are many research groups and methodologies to support these tasks, researchers in the area of software engineering still lack tools and methods that support the implementation of these activities and, in general, they need to use software that was not designed for this purpose. This paper presents an approach based on Biolchini's proccess, using checkpoints techniques, to assist in maintaining of the main objectives of the review process; these tasks were supported by a management software. The software facilitates the execution of repetitive tasks of recording, quantifying and classifying of data in accordance with a predefined research protocol in the planning phase, thereby enabling studies to be better organized and an overview to be obtained in the early stages of the review. We used a Systematic Review theme to validate the approach and supporting tool. This article shows that by visualizing and classifying research data while still at the initial stages of a Systematic Review, problems may be identified in the design of the protocol (planning phase), which otherwise would only be detected in the final stages, when results are being generated.**

Keywords- Systematic Review; Systematic Mapping; Support Tool.

I. INTRODUCTION

A Systematic Literature Review (SLR) is a way to identify, evaluate and interpret all relevant research available on a particular research question, study area or phenomenon of interest. Individual studies that contribute to a Systematic Review are known as the primary studies; a Systematic Review is a kind of secondary study [1].

The term Systematic Review is used to refer to a specific research methodology developed to obtain and evaluate evidence in a particular topic or research area [2]. In general, a Systematic Review involves three phases: *i)* Planning a review or developing a research protocol, *ii)* Running or conducting the review, and *iii)* Reporting the results [1]–[3].

The area of Software Engineering has shown interest in evidence-based studies where the presence of experimental software engineering is becoming more and more common in large events of the area [2][5][6].

The main problem tackled in this article is in tasks related to planning and conducting the research protocol, which require the researcher to be extensively dedicated and highly organized since he/she must catalog and classify the primary research and perform quantitative and qualitative analyzes, in order to get a broad view of the object of study which will facilitate the generation of results [5][6].

This paper proposes an approach based on the process proposed by Biolchini *et al*. [2] for the tasks of planning and conducting Systematic Reviews. A computer system stores the data progressively during the review and summarizes the results by each research protocol. The system seeks to reduce the time and effort needed for this process by eliminating the need to transfer information between various software programs, a situation that arises when these have not been designed to support the review process appropriately.

This article shows that by visualizing and classifying research data while still at the initial stages of a Systematic Review, problems may be identified in the design of the protocol (planning phase), which otherwise would only be detected in the final stages when results are being generated. In order to validate the approach and the software system, these were used to conduct a Systematic Review of a Masters Dissertation [9].

This paper is organized as follows: Section 2 presents the theoretical framework for Systematic Reviews in the field of software engineering and related studies; Section 3 presents the approach, Section 4 presents a case study where the approach and the system support were experimentally used and Section 5 presents the conclusions and final considerations.

II. SYSTEMATIC REVIEWS

This section presents the concepts that comprise the use of Systematic Reviews in the field of software engineering and related studies.

A. Systematic Reviews in the Context of Software Engineering

Any consolidated research area ends up producing a lot of papers and results, which require summarization and classification, therefore, enabling a broader understandig of

 ISBN: 978-1-61208-367-4

the field [3]. Although Systematic Reviews are widely used in medicine, they have only recently attracted the interest of researchers in software engineering [1]–[3].

In brief, the difference between Systematic Reviews and literature reviews can be seen in the way they are conducted. In a Systematic Review, there are a rigorous methodologies in which criteria for including or excluding a research study and steps for conducting the research are pre-specified through in a research protocol; unlike a literature review, which presents the studies analyzed without details of how they were chosen [2]. In literature reviews, there are no explicit explicit criteria for including and excluding studies in a review nor is other information given which wold enable a Systematic Review to be scientifically replicated or extended in a methodologically rigorous way.

The characteristic of Systematic Mappings is that they mainly focus on generating results in visual form, mapping itself being a particular area of research. On comparing Systematic Reviews and Systematic Mappings, it can be concluded that both involve the same methodological rigor, and they are often used, loosely, as synonyms. The main difference between them can be found in their goals and not in their methods [4].

Kitchenham [1] adapted the guidelines given to conduct reviews in medicine, including the best known one, "*The Cochrane Reviewer's Handbook*" [10], to the specific area of software engineering. Systematic Review were divided into three phases, namely, *i)* Planning, *ii)* Conducting, and *iii)* Reporting results. He also discusses the reasons for conducting a SRL in the area of software engineering:

- Summarize existing evidence about a technology.
- Identify gaps in current research in order to suggest areas for future research.
- Provide knowledge on new research activities.

The tasks related to the production of secondary studies by Systematic Reviews are carried out in three (3) distinct phases in several papers in the literature: In [5], a model, similar to those found in [1] and [2], was used. It is also performed in three phases, namely, developing a research protocol, conducting the research and reporting the results.

The model used by Montoni [5] was also used in the Systematic Review by Barcelos [6].

The phases of planning and implementing research which precede the step of generating the results, require greater manual effort by the empirical researcher. These phases involve defining of all the protocol items related to the research questions; the stages of selecting the criteria and excluding items. In addition, they include defining the sources in which searches for studies will be conducted, the primary language publications and the period the search will cover. The research has to strictly maintain the criteria defined in the protocol so as to avoid the search generating biased results.

In [2], Biolchini *et al*. presents a template for performing Systematic Reviews where the incremental use of the following process is recommended.

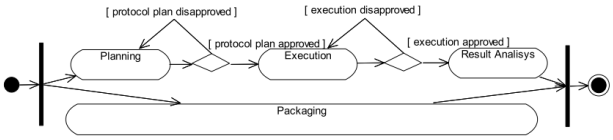


Figure 1. Proccess Overview [2].

The number of Systematic Reviews has grown insoftware engineering. The literature contains studies that propose solutions for problems found at various stages of the review process. Dieste *et al*. [7] gives strategies and methods for detecting relevant experiments, in particular draws attention to creating effective keywords for searches.

Montoni [5] and Barcelos [6], their implementation phase of the review, used a relational database to store information gradually with regard to title, author (s), year of publication, event, search source, classification and a brief summary of each article published is made while conducting the search so as to facilitate reaching the generation of the final results. Laguna and Crespo [8] used the *Mendeley* [12] software for managing data in review. Despite these studies having guidelines proposed by Kitchenham [1] in common, note that they use and adapt different tools and techniques for the same purpose, besides which their level of detail and the stages of the process are different in each study.

III. PROPOSED APPROACH

Based on three main steps, as in [1]–[3], wich comprise a Systematic Review, our approach proposes to conduct the review in compulsory and interdependent steps with the presence of checkpoints to guide inexperienced reviewers in a step-by-step style. We used a system for storing, retrieving and classifying information and tasks; that comprise a review, the *revision manager* [11].

The figure below shows an overview of the proposed methodology. A tool to support the process allows the insertion of checkpoints which enables the results of alignment with the objectives of the review and modification of the protocol for error correction to be aligned incrementally.

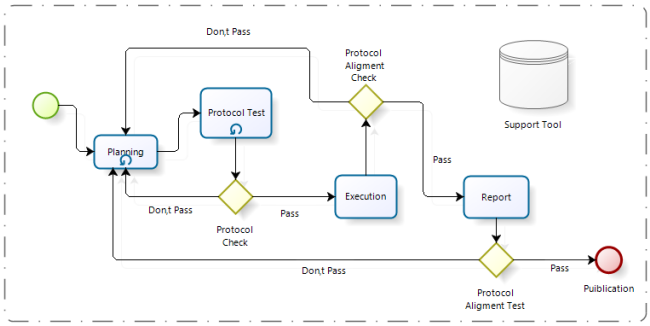


Figure 2. Proposed Approach [9].

The planning phase includes the construction of the research protocol itself, when its main items were defined. In [5], Montoni recommends defining the following items in the

planning phase: Context of the study, objectives to be achieved, research questions, research sources, languages, methods used in the search for primary publications, procedures and criteria for including studies, procedures for data extraction and analyzing results. It should also establish procedures and metrics for implementing the testing protocol.

The implementation phase is when the pre-defined protocol is followed so as to obtain primary research studies. Finally, we make a quantitative and qualitative analysis of the data in order to get an overview of the object of study and publish the results.

A tool was developed to support the phases of the Systematic Review. The Revision Manager (RM) [11], was developed using the Django web development framework [13], relational database MySQL [15] and a Javascript library for creating graphical reports, the Hightcharts [14].

The central idea of the tool is to avoid change in the working environment as well as to generate reports while still in the early phases, thereby enabling problems to be identified at an early stage.

IV. TOOL OVERVIEW

Currently in version 1.0, the Systematic Review Supporting Tool [11] is able to store and manage data of the review itself as well as information concerning the evaluation of each article, including information regarding the steps, research sources and selection criteria. This information is used to clustering and classify articles. The system is multi user, wich allows each user to access only data related to his/her own work. There is also the isolation of a review, where it is available only to the person who conducted the review.

Each stored item has a Create, Read, Update and Delete (CRUD) functionality to manage data thus allowing information regarding the protocol to be refined during the research. The figure shows the ERD (Entity Relationship Diagram) system.

Based on the process used, here is an overview of the workflow of the supporting tool:

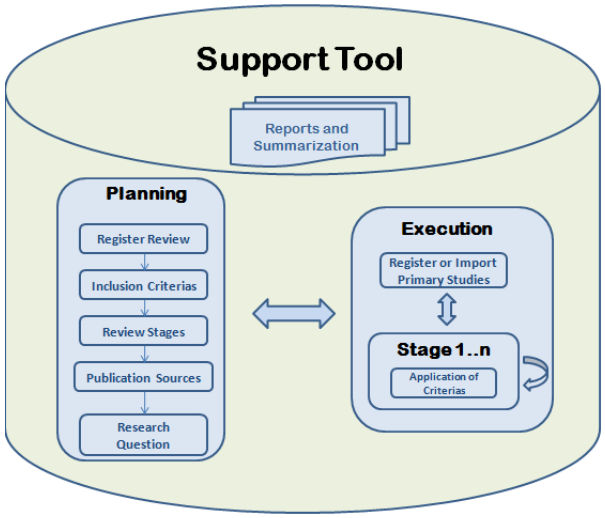


Figure 3. Tool Flow.

Figure 3 shows the incremental use of the tool by allowing reports to be generated at any point in the process. This procedure facilitated the identification of errors and the need to change the protocol.

Figure 4 reveals the presence of relationships which permit information of each step of the review to be stored and retrieved. This feature is essential so that reports can be generated and actions performed during the search traced.

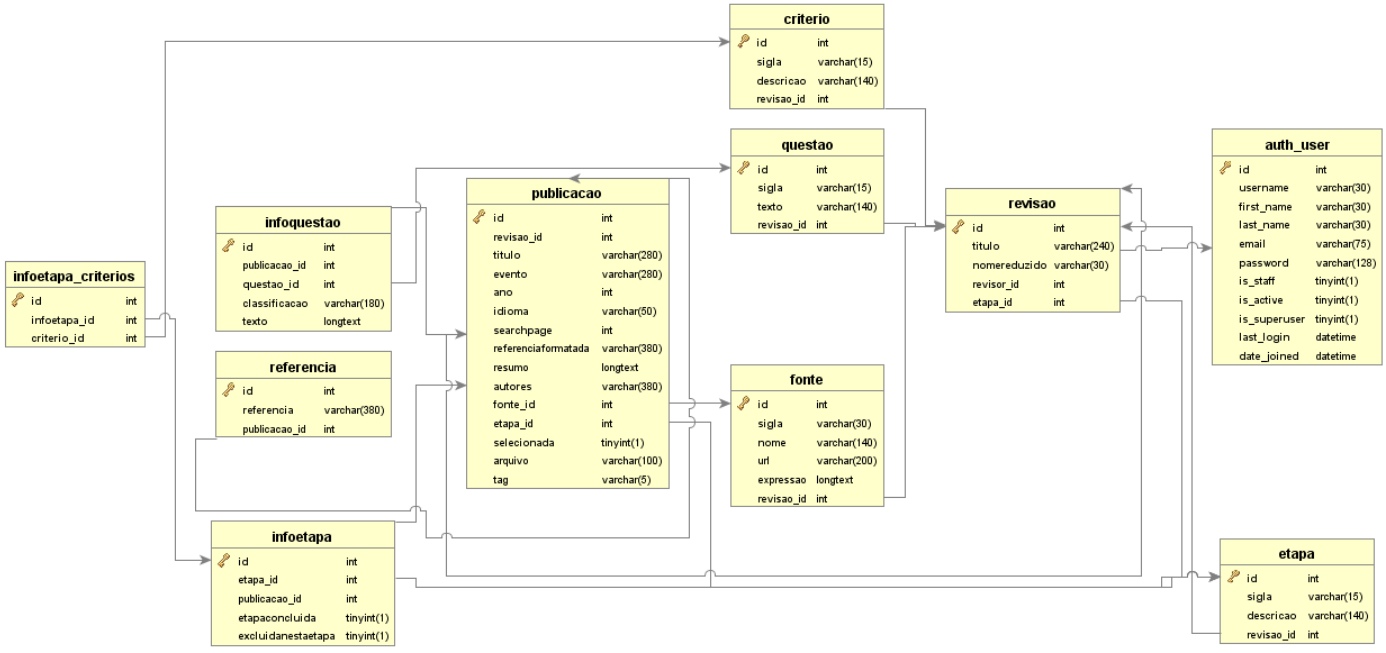


Figure 4. Entities and Relations Diagram.

V. USE CASE

We used the tool to support the management of a Systematic Review methodology of a final Master's project [9]. The planning phase followed the models of Systematic Reviews present in [1][2][5][6], was adapted for specific research items.

After proper registration in the review system, protocol data were stored and alignment criteria were specified. These criteria were evaluated at various points of the review. The alignment criteria are defined taking into account quantitative articles related to the theme and their relevance with regard to answering the research questions of the review.

In the planning phase of the protocol, it was decided that conduct phase would be divided into 3 stages (not to be confused with the general phases of the review). In each subdivision, a set of inclusion and exclusion criteria were applied to articles and every action was recorded by the system. The following is a summary of each subdivision:

a) Step 1 (E1): Obtaining articles by applying the search expression in research sources.

b) Step 2 (E2): Applying the selection criteria to the title and summary.

c) Step 3 (E3): Applying the selection criteria when reading the full text.

During the application of filters, checking every criterion was checked for alignment with the protocol as proposed in [6]. This action sought to identify if there was a need to refine the protocol. There follows a summary of the rules for alignment:

a) In relation to the number of articles before applying the filters: A high number may indicate that the search expression must be refined because it covers an area larger than the one desired. On the other hand, a very small volume may indicate the premature elimination of relevant publications.

b) In relation to the number of articles after applying the set of filters and selection criteria: A very high number or one very close to the number obtained in the early stages may indicate that there are unnecessary steps or that the criteria are too close to the original search expression.

The following is a summary of the results obtained at the beginning of the review:

TABLE I. SOURCES.

Source	Address
Compendex(CPX)	www.engineeringvillage.com
ScienceDirect(SCD)	www.sciencedirect.com

Table I shows the digital sources used. In the first iteration, we used the expression for searching databases. The period used as a search criterion was from January 2009 to January 2014. The results obtained are shown in Table II and Table III:

TABLE II. FIRST EXECUTION.

STAGE II			
Source	**Articles**	**Excluded**	**Approved**
CPX	152	50	102
SCD	483	433	50
Total	**635**	**483**	**152**

Due to the high rate of exclusions based on analyzing the abstract and title (E2), we opted not to proceed immediately to the next step, which comprises the full reading of the articles approved. We, therefore, interrupted the process in order to hone the search expression and selection criteria.

In the second iteration, after a terse expression search, and modifying the search period so that it ran from January 2010 to January 2014 and include new criteria for inclusion and exclusion the following quantitative data were yielded:

TABLE III. SECOND EXECUTION.

Stage (E1)			
Source	**Articles**	**Excluded**	**Approved**
CPX	94	0	94
SCD	188	0	188
Stage (E2)			
Source	**Articles**	**Excluded**	**Approved**
CPX	94	29	65
SCD	188	178	10
Stage (E3)			
Source	**Articles**	**Excluded**	**Approved**
CPX	65	36	29
SCD	10	10	0
Approved: 29			

Due to all articles obtained from *ScienceDirect* being excluded, we chose to remove it from the list of sources of the research protocol and to make refinements before starting a new implementation phase. Further details about the refinements and overall results can be found in [9].

VI. THREATS TO VALIDITY

The following two itens sumarises the main threats to the validity of the results of a this study:

(i) The proposed process was not measured, or no empirical experiments were conducted (apart from it being used in one case study). To further validate a process like this others factors should be considered as the number of articles in similar reviews. This can be used as a parameter.

(ii) The data in this study were not statistically analyzed with the standard measurements (mean, standard deviation, etc.) because the process was tested only once. Different iterations will help to understand the process and how it behaves.

VII. CONCLUSIONS AND FUTURE WORK

This paper presented an approach for conducting Systematic Reviews using checkpoints to check the alignment of the results obtained and expected. We used a

 ISBN: 978-1-61208-367-4

tool to support data management and to generate quantitative results in the early stages of the review.

Given the excessive number of manual and repetitive tasks that are involved in this type of research it is believed that this approach together with the software support proposed contributes in particular to helping researchers who have no experience in Systematic Reviews. The proposed approach enable the researchers to focus their efforts on tasks related to qualitative empirical analysis while the quantitative analysis and classification is performed by the software support system.

We intend to provide the system and its user manuals available for use by the academic community. We also intend to validate it by conducting other reviews. We hope that users will add features they require, thereby contributing to the improvement of the system and the approach.

REFERENCES

[1] B. Kitthenham, Procedures for Performing Systematic Reviews. Technical Report TR/SE-0401: Keele University and National ICT Australia Ltd, 2004, p. 33.

[2] J. Biolchini, P. G. Mian, Ana Candida Cruz Natali, and G. H. Travassos, Systematic Review in Software Engineering, no. May. Technical Report RT - ES 679/05: COPPE/UFRJ/Programa de Engenharia de Sistemas e Computação/Rio de Janeiro-RJ, 2005, p. 31.

[3] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," 12th Int. Conf. Eval. Assess. Softw. Eng., pp. 71–80, 2008.

[4] F. Q. B. da Silva, M. Suassuna, R. F. Lopes, T. B. Gouveia, a. C. a. Franca, J. P. N. De Oliveira, L. F. M. De Oliveira, and A. L. M. Santos, "Replication of Empirical Studies in Software Engineering: Preliminary Findings from a Systematic Mapping Study," 2011 Second Int. Work. Replication Empir. Softw. Eng. Res., pp. 61–70, Sep. 2011.

[5] M. A. Montoni, Uma investigação sobre os fatores críticos de sucesso em iniciativas de melhorias de processos de software. UFRJ/COPPE/ Programa de Engenharia de Sistemas e Computação, Rio de Janeiro - RJ: Tese (Doutorado), 2010, p. 400.

[6] M. P. Barcelos, Uma estratégia para medição de software e avaliação de bases de medidas para controle estatístico de processos de software em organizações de alta maturidade. UFRJ/COPPE/ Programa de Engenharia de Sistemas e Computação, Rio de Janeiro - RJ: Tese (Doutorado), 2009, p. 433.

[7] O. Dieste and A. G. Padua, "Developing Search Strategies for Detecting Relevant Experiments for Systematic Reviews," First Int. Symp. Empir. Softw. Eng. Meas. (ESEM 2007), pp. 215–224, Sep. 2007.

[8] M. a. Laguna and Y. Crespo, "A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring," Sci. Comput. Program., vol. 78, no. 8, pp. 1010–1034, Aug. 2013.

[9] M. G. da S. Neto, Uso de SOA na modernização de Softwares Legados: Um Mapeamento Sistemático da Literatura. CESAR-edu/MPES/Recife-PE: Dissertação (Mestrado), 2014, p. 163.

[10] Cochrane Collaboration. Cochrane Reviewers' Handbook. Version 4.2.1. December 2003

[11] M. G. da S. Neto, (retrieved: 2014, August 20). Review Manager Source Code [Online]. Available: https://github.com/academicsgit/reviewmanager

[12] Mendely Inc., (retrieved: 2014, August 20). Mendeley Webpage [Online]. Available: http://www.mendeley.com

[13] Django Home, (retrieved: 2014, August 20). Django Framework Webpage [Online]. Available: http://www.djangoproject.com

[14] Hightcharts Home, (retrieved: 2014, August 20). Hightcharts Webpage [Online]. Available: http://www.hightcharts.com

[15] MySQL Inc., (retrieved: 2014, August 20). MySQL Webpage [Online]. Available: http://www.mysql.com

 ISBN: 978-1-61208-367-4

Productivity-Based Software Estimation Model: An Economics Perspective and an Empirical Study

Alain Abran, Jean-Marc Desharnais
ETS, University of Québec
Montreal, Canada
alain.abran@etsmtl.ca
jean-marc.desharnais@etsmtl.ca

Mohammad Zarour
KAST
Saudi Arabia
mzarour@kacst.edu

Onur Demırörs
Middle East Technical University
Ankara, Turkey
demirors@metu.edu.tr

***Abstract*— Management interest is not limited to accurate estimate of software projects, but also to being more productive than your peers. This paper proposes an estimation approach based on economics concepts, such as productivity models with fixed/variable costs and economies/diseconomies of scale. This paper also reports on an empirical study in a Canadian organization that illustrates this approach.**

Keywords-Software economics; productivity models; fixed and variable cost; Function Points.

I. INTRODUCTION

Over the past 40 years, researchers have tackled software effort estimation using different mixes of cost drivers as well as various techniques to combine these costs drivers using either expert opinions or mathematical models: their main goal is to come up with 'accurate estimates', either intuitively based on experts opinions, or through mathematical models, derived from distinct broad strategies for designing estimation models:

- Strategy 1: Statistical analyses taking into account only the information from completed projects. They are represented by multi-variable models with as many independent variables as there are cost drivers taken into account. Some examples are: linear and nonlinear regressions techniques, neural network models, and genetic algorithms. For an adequate statistical analysis, it is generally accepted that there should be 20 to 30 observations for each independent quantitative variable.
- Strategy 2: Statistical analyses with a unique independent variable (typically, size) combined with a single adjustment combining the impact of multiple cost drivers, which individual values come from fixed pre-determined step-functions for each cost driver. This can be observed, for instance, in the COCOMO-like models [1][2].

Multi variables models built with insufficient data points (strategy 1) or with models with an adjustment factor bundling multiple categorical variables (strategy 2) do not necessarily reduce the risks inherent in estimation: they may lead managers to believe that the majority of important cost drivers have been duly taken into account by the models but, in practice, even more uncertainty has been created.

Although accurate estimation of a single project is important, estimation is not the unique management concern, nor the most important one for a specific project or for a set of projects for an organization or a customer. For example, greater productivity, profitability, and high quality have often greater management relevance than accuracy of estimates. In contrast to the traditional approaches in software engineering focusing strictly on estimation, this paper looks at an approach common in economics which looks first at productivity, a single variable model, before moving on to multi-variable models for estimation purposes in specific contexts. Some of the concepts introduced in this paper have been explored initially in [3] to identify a new approach to software benchmarking and estimation. This paper expands on these concepts and reports on an empirical study that illustrates the contribution of these concepts from economics in developing tailor-made estimation models based on the performance of the organization studied.

The rest of this paper is organized as follows. Section II presents the productivity concept as defined in economics to represent the performance of a production process, including fixed/variable costs and economies/diseconomies of scale. Section III presents the context of an empirical study in a Canadian organization. Section IV presents the productivity analysis and the estimation models developed for this organization on the basis of economic concepts. Section V presents a summary and implication for estimation purposes.

II. PRODUCTIVITY MODELS AND ECONOMICS CONCEPTS

A. A productivity model represents a 'production' process

A project, on the one hand, is typically set up to plan and manage an unique event, with a start date, an end date, and a unique outcome.

On the other hand, to improve the odds of meeting the project targets, a project process is implemented to plan activities, monitor project progress, and take remedial action when something goes off track. Similarly, even though each piece of software is different, its delivery is organized in a structured manner and not left to randomness and individual moods and intuitions of the day: to deliver the right outcome on time and within the expected cost and level of quality, a 'development process' is implemented to meet the target taking into account the set of priorities and within a reasonable range of predictability.

The question is: How can the performance of a process be estimated in the future if its current and past performance

 ISBN: 978-1-61208-367-4

and any variations in performance are not known? What are the economic concepts at work in software projects? And, when this is understood and quantified, how can these economics insights be used for estimation purposes?

A software development project can be modeled as a production process in its simplest form, with three main components:

1) Inputs: to calculate productivity, the people involved in the production process are considered as the inputs from an economics perspective. In a software project, the inputs are typically measured in work-hours (or person-days/-weeks/-months).
2) Activities within the process itself: for productivity calculation, all of the activities and constraints of the process are considered as a black-box and are not taken into account: they are therefore implicit variables, not explicit variables in productivity calculations.
3) Outputs: the outputs are represented by the number of functional units produced by the process. In a car manufacturing plant, the outputs of the plant are the number of cars produced (not the technical characteristics of the car, such as the weight, colors, shape, etc.). In comparison, the output of the software development process is the set of functions delivered to the users, which functions can now be quantified with international standards of measurements, such as with anyone of the relevant ISO standards on software functional size [4][5][6][7].

The productivity of a process is its ratio of outputs over the inputs used to produce such output. In software, the productivity of a software project can be represented as 10 Function Points per work-month. It is to be observed that, by convention, the productivity ratio ignores all process characteristics: it is process and technology independent and allows therefore objective comparison of the productivity of a process across technologies, organizations and time.

B. Productivity models with fixed and variable costs

The use of productivity models has a long history that can be traced back to a large body of knowledge developed in the domains of economics and engineering [8][9]. This section introduces some of these concepts which may also be useful in modeling, analyzing and estimating the performance of software projects.

A productivity model is typically built with data from completed projects, that is, it uses the information of a project for which there is no more uncertainty on:

- The outputs: i.e., all the software functions have been delivered; and,
- The hours worked on the project: i.e., they have been accurately entered into a time reporting system.

This illustrated in Figure 1 where:

- The x axis represents the functional size of the software projects completed;
- The y axis represents the effort in number of hours that it took to deliver a software project.

The straight line across Figure 1 represents a statistical model of the productivity of the software projects. More specifically, this single independent variable linear regression model represents the relationship between effort and size, and is represented by the following formula:

Y (effort in hours) = f (size)

= a x Size + b where:

- Size = number of Function Points (FP)
- a = variable cost = number of hours per function point (hours/FP)
- b = constant representing fixed cost in hours

In terms of units, this equation gives:

Y (hours) = (hours/FP) x FP + hours = hours

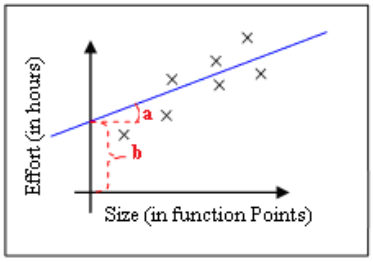


Figure 1. Fixed and variable cost in a productivity model

Insights from economics have identified two distinct types of costs incurred to produce different quantities of the same types of outputs:

Fixed costs: the portion of the resources expended (i.e., inputs) that does not vary with an increase in the number of outputs. In Figure1, this corresponds to b, the constant in hours at the origin when size = 0.

Example of a *fixed* cost: a cost of b hours of project effort is required for mandatory project management activities, whatever the size of the software to be developed.

Variable costs: the portion of the resources expended (i.e., inputs) that depends directly on the number of outputs produced. In Figure 1, this corresponds to the slope of the model, that is: slope = a in terms of hours/FP (i.e., the number of work hours required to produce an additional unit of output).

It is to be observed that in productivity models, the constant b does not represent the errors in the estimates as in multi-variable estimation models: in productivity models, it has a practical interpretation corresponding to the economics concepts explained above, that is: the portion of the cost that do not vary with increases in the production outputs.

C. Economies and diseconomies of scale in productivity

In economics, various behaviors in productivity have been observed as the number of outputs increases. For instance, that are some processes where:

- As output increases, 1 additional unit of output requires a smaller increase of inputs, and

 ISBN: 978-1-61208-367-4

- As output increases, 1 additional unit of output requires a greater increase in input.

When the increase in output units requires a correspondingly smaller increase in the number of input units, the production process is said to have lower sensitivity to size increases: this is referred to as 'economies of scale' (i.e., the larger the number of units produced, the more productive the production process).

By contrast, when an increase in output units requires a larger increase in the number of units for each additional output, then the production process is said to have diseconomies of scale (i.e., it is highly sensitivity to increases in size: for each additional unit produced, the less productive the production process = diseconomies of scale).

The next question is, of course, what cause these different behaviors? Of course, the answers cannot be found by graphical analysis alone, since in productivity there is only a single independent quantitative variable in a two-dimensional graph. This single independent variable does not provide, by itself, any information about the other variables, or about similar or distinct characteristics of the completed projects for which data are available. Efficiency investigation with additional independent variable can help identify which other variables cause variations in productivity and to which extent for each.

When a data set is large enough (that is, 20 to 30 data points for each independent variable), the impact of the other variables can be analyzed by statistical analysis. In practice, most software organizations do not have data set large enough for valid multi-variable statistical analysis. However, within a single organization the projects included within a data set can be identified nominally by the organizations that collected the data [3][10]. Each project in each subset should be analyzed next to determine:

- Which of their characteristics (or cost drivers) have similar values within the same subset; and
- Which characteristics have very dissimilar values across the subsets.

Of course, some of these values can be descriptive variables with categories (i.e., on a 'nominal' scale type: for example, a specific Data Base Management System (DBMS) has been used for a subset of projects, etc.). It then becomes necessary to discover which additional independent variables have the most impact on the relationship with project effort. The different values of such characteristics can then be used to characterize such datasets, and for selecting which of these productivity models to use later on for estimation purposes.

III. A PRACTICAL USE OF THESE ECONOMIC CONCEPTS: AN EMPIRICAL STUDY

A. Context

A Canadian organization was interested in determining its own productivity, in understanding some of the key drivers behind its major productivity variations, and in using the findings to improve its estimation process.

This organization, a government agency, provides specialized financial services to the public, and its software applications are similar to those of banking and insurance providers. It has a software development methodology fully implemented across all of its projects. The main objectives of this empirical study were to:

1. Internal benchmarking, i.e., compare the productivity of individual projects.
2. Develop estimation model(s) based on the data collected.
3. Identify and explain significant productivity variations across their projects.

B. Data collection procedures

The initial step was to identify the projects that could be measured for the productivity and benchmarking analyses. The selection criteria were:

- Projects completed within the previous two years, and
- Project documentation available for functional size measurement.

For this study, all data were recorded using the data field definitions of data collection questionnaire of the International Software Benchmarking Standards Group [11] [12].

C. Data Quality Controls

Quality control of the data collection process is important for any productivity study. Here, two quantitative variables are critical: the effort reported for each project, and the project functional size:

A- Effort data: in this organization, the time reporting system is considered highly reliable and is used for decision making, including payment of invoices when external resources are hired to complement project staffing.

B- Measurement of functional size: the quality of the measurement results depends on the expertise of the measurers and on the quality of the documentation available for the measurement process. For this productivity study, all functional size measurements were carried out by the same measurer with years of 20 years expertise in both functional size measurement methods used.

D. Descriptive Analysis

For this study, the 16 software development and improvement projects completed between 2004 and 2006 were measured in terms of functional size, effort, and various environment qualifiers. The staff who developed

these projects included both internal and external developers, distributed equally overall. In summary:

- Project sizes vary from a minimum of 111 FP (project 7) to a maximum of 646 FP (project 2).
- Effort varies from 4,879 hours to 29,246 hours.
- Unit effort varies from 14 hours/FP for project 12 to up to 98 hours/FP for project 6, a factor of approximately 8 between the least productive and the most productive within the same organization.
- Duration varies from 10 to 35 months.
- Maximum development team sizes for 12 of the 16 projects were available, and ranged from 6 to 35 employees.

The descriptive statistics of this dataset are as follows:

- Average effort = 12,033 hours (or, 1,718 person-days at 7 hrs per day, or 82 person-months at 21 days per month).
- Average unit effort is 41.5 Hrs/FP
- Average duration = 18 calendar months.

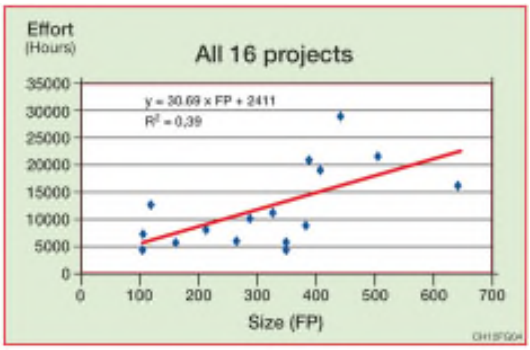


Figure 2. The organization's overall productivity model – N = 16 projects

IV. PRODUCTIVITY ANALYSIS AND ESTIMATION MODELS

A. *The overall productivity model for this organization*

The dispersion of points for this organization is illustrated in Figure 2 for all 16 projects, with functional size on the x axis, and effort on the y axis: it shows also the overall single-variable productivity model for this organization, using a single regression model:

Effort = 30.7 hours/FP x project size + 2,411 hours

The coefficient of determination (R^2) of this model is relatively low, at 0.39.

The practical interpretation of the above equation is as follows for this organization:

- Fixed effort = 2,411 hours
- Variable effort = 30.7 hours/FP

The possible reasons for the rather high fixed and high variable unit effort figures have been discussed with the managers, and the following observations provided in terms of the development methodology deployed in this organization:

A. It is highly procedural and time-consuming;
B. It included heavy documentation requirements;
C. It requires lengthy consensus building procedures across stakeholders and development staff;
D. It requires a relatively high number of inspections.

From Figure 2, it can be observed that, for this organization, 5 projects have costs 100% higher than projects of comparable functional size:

- Project with 126 FP required twice as much effort as 2 other projects of similar size.
- Four large projects (between 400 and 500 FP) required two or three times as much effort as similarly sized projects: the effect of these projects is to pull up the linear model (and corresponding slope) and to influence both the fixed and variable costs considerably.

This data sample was therefore split into two groups for further analysis.

A. A group of 11 projects which have the best productivity (i.e., lower unit effort, and which are below the regression line in Figure 2).
B. A group of 5 projects which have a much worst productivity (i.e., a unit effort twice the unit effort of the 11 other projects, and which are above the regression line in Figure 2).

B. *Organizational process capability: the most productive projects*

A productivity sub-model is built next with the 11 projects with a much lower unit effort per project that is, the most productive ones. For these projects, the linear regression model is:

Effort = 17.1 hours/FP x size of the project + 3,208 hours

The coefficient of determination (R^2) of this model is 0.589, higher, relatively, than that for the overall model.

The practical interpretation of this equation is:

- Fixed costs = 3,208 hours
- Variable Costs = 17.1 hours/FP

C. *Productivity model of the least productive projects*

Another productivity sub-model is built with the 5 least productive projects in group B. For these projects, the linear regression model is:

Effort = 33.4 hours/FP x project size + 8,257 hours

The coefficient of determination (R^2) of this model is better, at 0.637. Of course, with a sample of only five projects, this number is not statistically significant, but is still interesting for this organization.

The practical interpretation of the above equation is as follows:

- Fixed effort = 8.257 hours
- Variable effort = 33.4 hours/FP

This group of the five least productive projects is characterized by a fixed cost which is almost 4 times higher than that of the full set of projects (8,257 hours vs. 2,411 hours), and a relatively similar variable effort unit (33.4 hours/FP vs. 30.7 hours/PF).

 ISBN: 978-1-61208-367-4

The group of 11 most productive projects is characterized by a fixed cost which is approximately 40% lower than that of the least productive projects (3208 hours vs. 8257 hours), and a variable unit effort which is almost 50% lower (17.1 hours/FP vs. 32.4 hours/FP); that is, with interesting economies of scale and an R^2 of 0.55.

A summary of each group is presented in Table I, where these 11 projects represent the organization's 'capability' to deliver in normal conditions and the other five projects illustrate how projects are significantly impacted in the presence of factors which have not yet been identified through this single independent variable (i.e., functional size) analysis. Exploration of these additional impact factors is discussed in Section V.

D. Qualitative causal analysis

Of course, a single independent variable model cannot explain the causes of such variations. Furthermore, there are often not enough data points within a single organization (unless they have been collecting data for many years) to rely on quantitative analysis with a dataset of only sixteen projects: each additional independent typically requires 20 to 30 additional data points. In the absence of sample sizes large enough for quantitative analysis, qualitative analysis can help identify probable causes of increases. In the context here, qualitative analysis will not attempt to quantify precisely the impact of a cause (or cost drivers), but will attempt to identify qualitatively which factor could have had the greatest negative impact on productivity.

TABLE I. FIXED AND VARIABLE EFFORTS: CAPABILITY VERSUS LEAST PRODUCTIVE PROJECTS

Samples/ Regression coefficients	**All 16 projects**	**Most productive: 11 projects**	**Least productive: 5 projects**
Fixed effort (hours)	2,411	3,208	8,257
Variable effort (hours/FP)	30.7	17.1	34.4

Off hand in the causal analysis of the productivity variations in this organization, two candidate cost drivers were eliminated since they were considered as constant in both groups of productivity performance:

- Development methodology: in this organization the use of the industry-tailored development methodology is fully deployed across all software development projects: none of the activities and controls can be bypassed. Therefore, there was no development methodology difference across all projects.
- Project managers' expertise: some of the projects managers had, within this same 2-year period, supervised projects which were both among the most productive and the least productive. Therefore the project management expertise of specific project managers could not explain large project productivity differences.

The question is, what are the factors that led to such large (i.e., +100%) increases in unit effort? What could have been the major cause-effect relationships? To identify and investigate these relationships, available project managers were interviewed to obtain their feedback on what they believed had contributed to either an increase or a decrease in the productivity of their respective projects. The project managers interviewed had managed 7 of the 16 projects:

A. 3 projects with the lowest productivity;
B. 2 projects with average productivity;
C. 2 projects with the highest productivity.

The aim of the interviews was to obtain qualitative information from the project managers on the factors they believed had contributed, or not, to the increase in project effort compared to that of other projects of similar size developed in the organization's environment or elsewhere during their project management practice. Their feedback is summarized in the following factors:

A- The most productive projects had the following characteristics:

1. Users familiar with both the business and software development processes;
2. Users involved throughout the project;
3. Software developers working on the projects who were experienced in the use of the development environment.

B. The least productive projects had the following characteristics:

B1. Customer related issues:

1. Customer requirements that were poorly expressed, or a customer representative who did not know his environment (business area), leading to frequent change requests during a project life cycle.
2. High turnover of users involved in the projects, leading to instability in the requirements and delays in decision making.
3. Customers not familiar with the software development process in the organization, including their required involvement in project activities, including activity reviews.

B2. Project constraints:

1. Tight project deadlines for legal constraints or public face-saving that led to compressed schedule and resources being piled up to make the problem disappear.
2. New technologies unknown to the developers.

B3: Product constraints:

1. Multiple links with other software applications of the organization.

An example of negative product constraint was reported for the project with the highest unit effort (98 hours/FP): the software delivered by this project was of a small functional size, but required twice as much effort to develop as another

 ISBN: 978-1-61208-367-4

of software of similar size because it interacted with almost all the other software applications of the organization and was dependent on other organizational units. Another project had a very tight deadline, which led management to 'throw' resources at the problem to meet the deadline irrespective of the total effort required.

It can be observed that, although it was possible to identify 'qualitatively' some factors with major negative impact, the sample size was much too small for statistical tests to quantify such an impact.

V. SUMMARY AND IMPLICATIONS FOR MANAGEMENT AND ESTIMATION PURPOSES

Taking into account the related performance concepts from the field of economics, including fixed/variable costs and economies/diseconomies of scale, this paper has reported on the productivity analysis of software projects developed by a governmental organization. For this organization, three productivity models were identified which represented respectively:

- An overall productivity of this organization. This overall productivity model will be used later across times periods to verify whether or not the productivity of this organization is improving over time, and with respect to external similar organizations.
- A productivity model built from the best productive projects: it exhibit economies of scale in the development process of this organization and represents its capability to deliver a software project with a lower fixed/variable effort structure, in the absence of major disruptive factors.
- A productivity model based on the five projects with the highest unit effort: in this organization, the presence of disruptive factors has led to greater than 100% increase in project effort in comparison to their organizational process productivity capability.

Of course, the limited number of projects available in these mathematical models does not permit generalization to other contexts, but it is describing quantitatively and objectively productivity facts: these models are representative of the organization studied in which a unique software development methodology is widely implemented and represents well deployed corporate software practices, not varying individual practices (i.e., a repeatable process rather than unpredictable individual and ad-hoc practices).

For estimation purposes, the organization's process capability model represented by the best performing projects should be used, provided that a risk analysis has not detected the presence of any of the disruptive factors that have in the past increased effort twofold in this organization. Whenever such disruptive factors are identified with a high probability of occurrence within an estimation context, it justifies this organization to estimate such projects using the productivity model derived from the least productive projects. The use of these two single-variable productivity models would be expected to provide more accurate estimates that the overall productivity model combining all previous projects.

In addition, interviews with project managers allowed to identified, qualitatively for this specific organization, factors having impacted, positively or negatively, productivity, (such as: customer related issues, project constraints and product constraints): these factors were integrated next as risk factors into their revised estimation process.

This context of an organization having measured only a small set of projects is representative of many organizations without much historical data: this is a context where there are not enough data points to build with high confidence multi-variable estimation models representing local conditions and related organizational performance.

The insights from productivity models developed from an economic perspective are important since relevant improvement actions may directly impact the productivity of the organization, by lowering either of the fixed or variable project costs.

REFERENCES

[1] B. W. Boehm, Software Engineering Economics, Prentice Hall, Englewood Cliffs, NJ, 1981.

[2] B. W. Boehm *et al.* Software Cost Estimation with COCOMO II, Prentice Hall, 2000.

[3] A. Abran and J. J. Cuadrado, "Software Estimation Models & Economies of Scale," 21st International Conference on Software Engineering and Knowledge Engineering - SEKE'2009. Boston (USA), July 1-3, 2009, pp. 625-630.

[4] ISO/IEC 19761: Software Engineering – COSMIC - A Functional Size Measurement Method. International Organization for Standardization, Geneva, 2011.

[5] ISO/IEC 20926: Software Engineering - IFPUG 4.1 Unadjusted functional size measurement method - Counting Practices Manual. International Organization for Standardization, Geneva, 2009.

[6] ISO/IEC 24750: Software Engineering - NESMA functional size measurement method version 2.1 - Definitions and counting guidelines for the application of Function Point Analysis, International Organization for Standardization, Geneva, 2005.

[7] ISO/IEC 20968: Software Engineering - Mk II Function Point Analysis - Counting Practices Manual. International Organization for Standardization, Geneva, 2002.

[8] N.G. Mankiw, Principles of Microeconomics, Ed. South-Western Cengage Learning, Mason (OH) 2014.

[9] S.T. Hackman, Production Economics - Integrating the Microeconomic and Engineering Perspectives, Springer-Verlag Berlin Heidelberg, 2008.

[10] A. Abran, I. Silva and L. Primera, "Field Studies Using Functional Size Measurement in Building Estimation Models for Software Maintenance," Journal of Software Maintenance and Evolution: Research and Practice, vol. 14, 2002, pp. 31-64.

[11] ISBSG R11, International Software Benchmarking Standard Group, 2009 URL: http://www.isbsg.org/ last accessed date: July 26, 2014.

[12] L. Cheikhi, A. Abran, and L. Buglione "ISBSG Software Project Repository & ISO 9126: An Opportunity for Quality Benchmarking," UPGRADE vol. VII, no. 1, 2006, pp. 46-52.

 ISBN: 978-1-61208-367-4

Measuring a Software Production Line with IFPUG-based Function Points

Volkan Halil Bagci, Umut Orcun Turgut, Ali Ciltik, Semih Cetin, Recep Ozcelik
Cybersoft Information Technologies R & D Center
Istanbul, Turkey
e-mail: {volkan.bagci, umut.turgut, ali.ciltik, semih.cetin,recep.ozcelik}@cs.com.tr

***Abstract* – Software Production Lines (SPLs) aim to manage cost-based activities for product delivery. Our company has been using SPL engineering for about 10 years and successfully implemented cost-controlled production cycles for SPLs during past two years, which are based on well-known Function Point (FP) approach supported by International Function Point User Group (IFPUG). Cost-based product delivery in SPLs requires the complete transformation of requirements gathering, cost estimation, time planning and productivity measuring steps. At the maturity level reached so far, every contributing part of the production line can be measured and cost-attached effectively and new targets can be set accordingly. Moreover, production bandwidth can be estimated precisely based on statistical productivity coefficients of every working team. This paper introduces our cost-controlled SPL approach, the achievements so far and our future plans for improvement.**

Keywords-Function Point; Software Measurement; Software Production Lines; Productivity Coefficient.

I. INTRODUCTION

Software is encountered in every part of our daily life nowadays. Consistent and cost-effective software products certainly make our life much easier. The consistency and cost-effectiveness of any product can be controlled by strict measurements and software is not an exception in that sense. In other words, software engineering is not an appropriate term unless the size, quality and productivity are measured accurately since unmeasured variables cannot be managed in any engineering discipline [1].

Software measurement enables the estimation of team productivity and improvement of existing processes based on recorded productivity metrics. Many researchers focus on new metrics to measure productivity [2] while others analyze software team productivity efforts and make empirical assessments for evaluating measurement efforts in software companies [3][4].

In particular to SPLs, the factors that accelerate and prevent team productivity can be statistically determined and exploited to the maximum extend for setting feasible targets. The approach explained in this paper has been used for the past two years in the banking SPL of our company and particularly implemented for a mid-scale bank in Turkey.

The rest of this paper is organized as follows: Section II discusses about the related works. Section III provides an overview of the organization and roles. Section IV describes the function point and its standard in the context of software evolution. Section V introduces the cost estimation process that is currently being held in our company. This section also describes the results obtained in the last period. Section VI discusses about the future work so as to improve the processes as a whole. Finally, Section VII concludes this paper.

II. RELATED WORKS

In the last decade many cost estimation models for software production lines have been proposed. Some representative proposals are: [5][6][7][8] and [9]. Poulin [5] presented a reuse metric and economics model that utilizes systematic reuse method. Poulin's model has two parameters: the relative cost of reuse (RCR) and the relative cost of writing for reuse (RCWR). Using these two parameters Poulin calculates the costs of product line development, thus provides extensive insight for the economics of software production lines.

Clements, McGregor, and Cohen [6] proposed the structured intuitive model for product line economics (SIMPLE) a general-purpose business model that supports the estimation of the costs and benefits in a product line development organization.

Lamine, Jilani, and Ghezala [7] proposed a new software cost estimation model for product line engineering that is based on integrated cost estimation model for reuse in general and Poulin's model of product line engineering. New tool supporting the model is described along with UML presentation.

Nóbrega, Almeida, and Meira [8] proposed integrated cost model for product line engineering (InCoME). As well as a new model is introduced along with its case study with results, the paper highlights important factors to acquire an effective model in terms of cost-benefit.

Nolan, and Abrahão [9] mentions about the experiences gained by using of estimation tools for the software product lines. It is clearly stated that a model is not only used for estimating cost and schedule but also for estimating and validating risks and opportunities. Future discussions about how a new cost model should be built are given for projects represented as number of Lines of Code (LOC).

 ISBN: 978-1-61208-367-4

III. MOTIVATION

Software Production Line is the adapted version of an industrial product line for developing software product families with the vision of managing cost and time-to-market concerns, which are based on structured reusability techniques [12]. The main supplier has its own SPL infrastructure so-called Aurora that is used for the production of different product families ranging from banking to insurance and tax administration to Enterprise Resource Planning (ERP) [10][11].

The customer bank decided to outsource the development and maintenance of its own banking software to the main supplier over its sister company the main contractor. In this setup, the main supplier is the main banking products supplier for many banks including the customer bank, and the main contractor company is the main contractor for customizing and maintaining the main supplier's banking products, particularly for the customer bank.

In order to provide high quality services to the customer bank, the main supplier and the main contractor decided to have a new unit called Product Management Department (PMD) in their joint organization chart. New organization chart including the PMD is given in Figure 1. PMD has a sub-unit so-called Product Improvement Group (PIG), which is responsible for inspecting and improving banking products (called product restructuring) using modern software engineering techniques as well as implementing corrective actions on existing modules (called product refactoring). Another sub-unit in PMD is Production Planning Group (PPG), which is responsible for cost-estimation of new inquiries, planning implementation tasks, and monitoring the production cycles. The contract between the customer bank and the main contractor is based on FP and inquiries are implemented with FP-based cost. FP-based cost anticipates the cost model based on software product functionality.

Pricing a single FP is not a trivial task in contractual terms since buyer and supplier do have different point of views. In case of the customer bank, a well-known international consulting group worked both with buyer and supplier teams to set the price for an FP, based on existing implementation costs and pricing models [13]. Working timesheets were examined, hourly and daily efforts were calculated and an average cost for an FP has been determined. Additionally, the FP-based cost estimation approach and related formula have been double-checked by the consulting group. The approach has been monitored for a while in real cases and finally approved both by the customer bank and the main contractor.

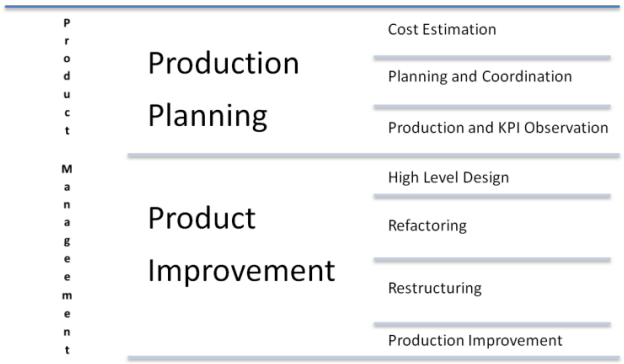


Figure 1. Organization schema of Product Management Department.

In this model, the customer bank Project Office (PO) only deals with the PPG as the single contact point of the main contractor and the main supplier. Once PO forwards inquiries, PPG prepares the Approximate Cost Form (ACF) for each inquiry, including the estimated starting and finishing dates of implementation. PO goes through each ACF and approves accordingly. The approval of ACF initiates the real planning of each inquiry with exact dates of implementation. PPG is also responsible for allocating necessary resources for the software development efforts. During the course of implementing every inquiry, PPG keeps certain Key Performance Indicators (KPI) to measure the effectiveness of every conveyor in the software production line. Using these SPL KPIs, PPG is expected to coordinate software development teams, business analysts, and test units throughout the lifecycle of a request.

IV. FUNCTION POINT

FP is a metric for measuring the functionality provided to the user of an information system. The concept was introduced by Albrecht in 1979 [14], and used widespread in the world as of today in a variety of 6 different standards, such as COSMIC FSM, FiSMA FSM, IFPUG FSM, MK II FPA, NESMA, and the automatic FP supported by Object Management Group (OMG) [15][16][17][18][19][20]. The OMG automatic FP standard is based on IFPUG approach in such a way that it determines functions, differentiates internal and external files, and calculates the FP accordingly.

IFPUG initiated the standardization of measuring software projects, which is accepted by the International Standards Organization (ISO) with most up-to-date version 4.3. As stated in IFPUG Counting Practices Manual (CPM) 4.3, FP is the unit of measurement to express the amount of business functionality [21]. IFPUG FP is calculated based on counting the factors, including internal and external information sources, external inputs, outputs, and queries. We particularly prefer to use IFPUG FP within other FP approaches as being the most widely used approach, being in line with banking domain, providing access to an extensive database of more than 5000 International Software Benchmarking Standards Group (ISBSG) project performance cases, having large volume of industrial data in management information systems, and enabling the official certification option [23][24].

 ISBN: 978-1-61208-367-4

V. COST ESTIMATION PROCESS

In this section of the study, cost estimation process is explained in detail while post process observations and outcomes are shared in later parts of the section.

Works that are being performed by the main contractor are handled via requests. Each request has a request type that might have impact on cost estimation as given in detail in Section A.

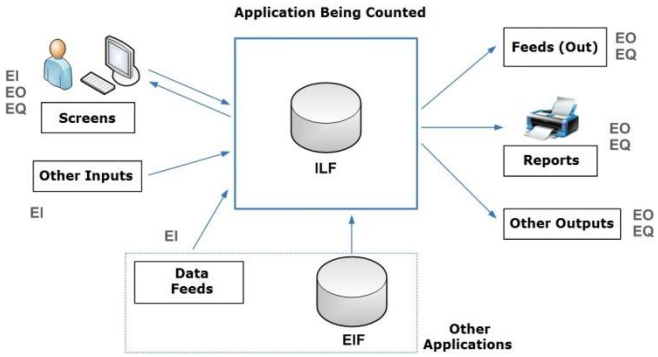


Figure 2. The view of a software application from the eyes of an FP practitioner [22].

Our FP practitioners examine requests to identify data and transaction functions using IFPUG FSM with similar view of a software application as shown in Figure 2. Once the initial examination of request is complete, project type and request size is decided as explained in section B and section C respectively. FP practitioners estimate cost of the request by using the process factor and calculation method as explained in section D.

After cost estimation is complete, planning and product phase starts as explained in Section E. Observations and importance of scope meeting are mentioned in Sections F and G respectively.

A. Request Type

Prior to our cost estimation process implemented, when a request is entered in the system, it is given a request type based on the expected application size and application type. In order to support these request types, general system characteristics (GSC) [21] are decided for these request types, making cost estimation balanced for a given type of request. Thus, there are three request types given in our system; which are project, improvement, and report.

Project and improvement types are both software applications that might involve brand new functionality and/or modifications over existing application. Main difference is the size of the application; for example, an estimated cost threshold of 62 FP or less is being used as improvement request type within our process. Any request that has estimated cost size of 62 FP is of project request type.

TABLE I. CALCULATED VAF VALUES FOR REQUEST TYPES.

Request Type	Total TDI Points	VAF (∑TDI * 0,01) + 0,65
Project	35	1.00
Improvement	35	1.00
Report	0	0.65

In our cost estimation process, request types can affect variable adjustment factor (VAF) as shown in (2), thus have impact on final cost estimation. VAF for project and improvement request types are set to 35 Total Degree of Influence Points (TDI), making VAF of these request types equal to 1.0.

$$VAF = (\sum TDI * 0.01) + 0.65 \text{ [21]} \quad (1)$$

Report is a special request type that addresses information retrieval using offline databases via quick third party development tools. VAF of report project type is calculated as 0.65 once all TDIs of the GSC are set to 0 due to the simple development efforts required for reports.

B. Request Requirements Category

Each request is represented by one or many requirements. These requirements can be identified as functional or non-functional ones. In our cost estimation process, while IFPUG FSM is used for functional requirements in terms of cost estimating, estimating cost of non-functional requirements handled using our non-functional point system. In order to cover a cost estimation process that would address requests with different possibility of requirement types, a request requirements category (RRC) is introduced as an element of decision node in our cost-estimation flow-chart, which is shown in Figure 3.

Based on the possible combination of the request requirement varieties, there are three RRCs as follows.

1) Functional RRC: Functional RRC addresses requirements that include only functional ones. Thus the cost of the request can be calculated according to IFPUG FSM v.4.3 standard. Whether the request has a functional component or not can be identified by examining the requirements of the request. If it has at least one function among Internal Logical File (ILF), External Interface File (EIF), External Input (EI), External Output (EO) or External Inquiry (EQ), then request may be processed as a functional RRC. Examples of functional projects are listed below.

- Data Migration (Customer data entrance, sending control signal)
- Data Transformation (Bank interest calculation, average temperature derivation)
- Data Storage (Customer order record, environment temperature record)

 ISBN: 978-1-61208-367-4

- Data Query (Listing current personnel, querying coordinate data)

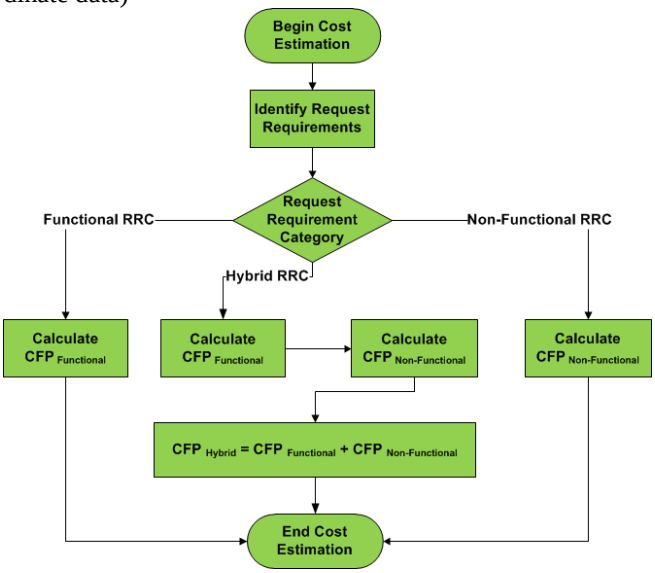


Figure 3. The cost estimation flow-chart.

2) Non-functional RRC: If the request does not contain a functional requirement, then the cost cannot be calculated using IFPUG FSM. The total cost of the request is calculated using non-functional point system by summing all of the separate FP costs of items, which are listed below in detail.

- Project Management, Coordination, Requirement Gathering
- Analysis, High Level Design, Quality Control
- Design, Software Development, Integration
- Functional Tests, Acceptance Tests, Technical Support Services
- Deployment, Fixed-Works

The costs of these types of requests are reckoned according to man/month data and used as NFP (Non-Function Point) in the system for setting connection with FP.

a) Fixed-Works: In order to decrease the operational cost of recurring non-functional requests, fixed-works list has been set up. Fixed-works list is a living document. Not only the FP practitioners, but also the planning experts and module managers do the relevant updates as NFP on that list.

3) Hybrid RRC: According to the standard of IFPUG FP calculation v.4.3, if cost price of a request can be executed, although it has non-functional requirements, this type of requests are called hybrid requests. The cost of these types of requests is calculated by summing the costs of both the functional and non-functional components.

C. Request Size

Requests that have a size below a certain threshold are classified as minor requests while the ones that are above the threshold are classified as major requests. Requests go through different states as shown in Table IV. Encompassing the period from request entry to the deployment, several output documents are created along these steps.

The aim of the request size classification is to have an efficient production line. As it can be seen in Figure 4 and 5, based on the request size, requests follow different path. With a few exceptions, minor requests usually get involved in a minor process pipeline, without passing through the analysis and design steps; thus, most of the documentation requirement is dropped off. On the other hand, major requests have to follow the big route, which is passing through quality processes and as a result, analysis and design documents are prepared in detail.

Current threshold in terms of FP is arranged to be just more than a single function, thus meaning if a request has more than a single function involved, it would be addressed as a major request. Based on IFPUG CPM 4.3 [21], minimum possible single function cost is 3 FPs; for example EI-Low and EQ-Low both have 3 FPs. Therefore, in agreement with the bank, it is decided to use 3 FPs as a threshold for request size classification.

D. Process Factor and Cost Calculation

Process Factor (PF) is the sum of total coefficients of all sub processes in the production line. It is used for reflecting the costs of all sub-processes to the total cost in minor and major requests. Besides distributing the total cost to the sub-processes, PF also calculates the partial cost when the job, which is being carried on the product line, is canceled. Maximum value for the PF can be 1.0. Table II details the PF values for some of the sub-processes and these are calculated according to their portions in the production period.

Equation (2) shows how the process factor is calculated.

$$PF = A + HLD + D + QC + DEV + AT + BT \qquad (2)$$

where A is the analysis process factor, HLD is the high level design process factor, D is the design process factor, QC is the quality control process factor, DEV is the development process factor, AT is the alpha test process factor and BT is the beta test process factor.

TABLE II. PROCESS FACTOR VALUES FOR EACH REQUEST TYPE.

Request Size / Request Type	A	HLD	D	QC	DEV	AT	BT	PF
Major Project	0.25	0.05	0.10	0.05	0.40	0.10	0.05	1.00
Major Improvement	0.25	0.05	0.10	0.05	0.40	0.10	0.05	1.00
Minor Improvement	0.00	0.05	0.00	0.02	0.40	0.10	0.05	0.62
Major Report	0.25	0.05	0.10	0.05	0.40	0.10	0.05	1.00
Minor Report	0.25	0.05	0.00	0.02	0.40	0.10	0.05	0.87

 ISBN: 978-1-61208-367-4

$$aFP = VAF * uFP \quad (3)$$

$$CFP = PF * aFP \quad (4)$$

As per definition given in [17] and shown in (3), adjusted function point (aFP) is calculated using VAF and unadjusted function point (uFP). Value of VAF can change based on the project type. As it can be seen from (4), PF has a direct consequence on the cost estimation. In (4), CFP is the cost in FPs, PF is the process factor and aFP is the regulated FP.

TABLE III. SAMPLE NET COSTS.

Request Size / Request Type	uFP	VAF Based on Request Type	aFP (VAF * uFP)	PF Based on Request Size	CFP (PF * aFP)
Minor Improvement	3.00	1.00	3.00	0.62	1.86
Major Improvement	32.00	1.00	32.00	1.00	32.00
Major Project	100.00	1.00	100.00	1.00	100.00
Minor Report	3.00	0.65	1.95	0.87	1.70
Major Report	6.00	0.65	3.90	1.00	3.90

With reference to the Table III, net CFP values for sample applications with distinct request types and request sizes are calculated according to distinct process factors. In the calculations, the threshold value is set to 3 FP.

E. Planning and Production Tracking

After estimating the cost of each request, planning experts decide on the deadline of the request, taking account of the characteristic of the request, source and integration status. Planned development time and the number of developers that are going to be assigned to the request are calculated according to the basic Constructive Cost Model (COCOMO) equations given in (7) and (8) [20]. In order to use these equations, reference values for planned development time and number of developers are calculated via (6) instead of (5). For this reason, instead of using code line of count parameter and COCOMO coefficients in (6), calculated effort value of product line is used and classical COCOMO equation is adapted to the (6) for our system. Since (5) is not being used directly, it does not have an effect on our productivity rates. c_b and d_b values are decided according to Boehm's semi-detached software project standards as stated in (7).

$$E = a_b (KLOC)^{b_b}, \; a_b = 3{,}0, \; b_b = 1{,}12 \quad [22] \quad (5)$$

$$E = \frac{CFP}{DM} \quad (6)$$

$$D = c_b E^{d_b}, \; c_b = 2{,}5, \; d_b = 0{,}35 \quad [22] \quad (7)$$

$$P = \frac{E}{D} \quad [22] \quad (8)$$

where E is the effort applied (person-months), KLOC is the estimated number of delivered lines of code for the project, a_b, b_b, c_b and d_b are COCOMO coefficients, CFP is the cost in FP, D is the development time in months, DM is the average work day count in a month (20 work days) and P is the count of required people.

TABLE IV. PRODUCTION LINE STATUS OF CORE BANKING UNIT.

	Production Line (FP)																				Amount
States	BKS		ÇEK		KYS		MEV		MUH		TAHSİS		TEM		TOTAL FP						#
01 Demand Admission	-	(0)	-	(0)	-	(0)	-	(0)	-	(0)	-	(0)	-	(0)	-	FP					8
02 Approximate Cost	-	(0)	-	(0)	-	(0)	-	(0)	-	(0)	-	(0)	-	(0)	-	FP					12
03 Approximate Cost Approval	0	(0)	0	(0)	297	(2)	0	(0)	0	(0)	0	(0)	0	(0)	297	FP					2
04 Planning	0	(0)	0	(0)	4	(2)	0	(0)	0	(0)	0	(0)	11	(1)	15	FP					3
05 Analysis	130	(4)	18	(3)	12	(4)	63	(5)	47	(1)	65	(3)	94	(6)	429	FP					26
06 High Level Design	0	(0)	0	(0)	24	(3)	0	(0)	0	(0)	19	(1)	0	(0)	43	FP					4
07 Quality Control 1	0	(0)	0	(0)	0	(0)	0	(0)	0	(0)	0	(0)	0	(0)	0	FP					0
08 Analysis Approval	6	(1)	0	(0)	115	(4)	0	(0)	15	(2)	114	(4)	121	(7)	371	FP					18
09 Design	4	(1)	40	(5)	7	(1)	83	(4)	0	(0)	0	(0)	48	(5)	182	FP					16
10 Final Cost	1	(1)	26	(1)	6	(1)	0	(0)	0	(0)	0	(0)	0	(0)	33	FP					3
11 Final Cost Approval	0	(0)	0	(0)	0	(0)	114	(1)	0	(0)	0	(0)	0	(0)	114	FP					1
12 Software Development	57	(4)	10	(3)	123	(3)	104	(9)	0	(0)	30	(3)	49	(8)	373	FP					30
13 Alpha Test	61	(2)	1	(1)	51	(4)	88	(6)	12	(2)	17	(1)	6	(1)	236	FP					17
14 Quality Control 2	0	(0)	0	(0)	7	(1)	23	(3)	0	(0)	0	(0)	0	(0)	30	FP					4
15 Quality Control	0	(0)	0	(0)	1	(1)	0	(0)	0	(0)	0	(0)	1	(1)	2	FP					2
16 Acceptance Test	20	(2)	4	(1)	49	(4)	52	(5)	20	(2)	61	(4)	10	(2)	216	FP					20
17 Acceptance Test Failed	0	(0)	0	(0)	0	(0)	0	(0)	0	(0)	0	(0)	0	(0)	0	FP					0
18 PO Control	0	(0)	10	(2)	0	(0)	12	(2)	0	(0)	0	(0)	1	(1)	23	FP					5
19 Deployment	218	(23)	87	(6)	76	(8)	285	(30)	6	(4)	88	(12)	93	(16)	853	FP					99
20 Production	0	(0)	0	(0)	1	(1)	0	(0)	0	(0)	0	(0)	0	(0)	1	FP					1
21 Completed	73	(20)	328	(61)	130	(30)	534	(83)	18	(11)	77	(17)	345	(64)	1.505	FP					286
22 Waiting	0	(0)	38	(1)	33	(1)	139	(3)	0	(0)	0	(0)	0	(0)	210	FP					5
23 Closed	4	(2)	0	(1)	18	(6)	0	(0)	2	(1)	0	(1)	34	(5)	58	FP					16

In order to obtain production line status data as show in Table IV, costs of requests are distributed among the request states. The production line status data enables us to track the current intensity of work load on each group and also to foresee the upcoming intensity of work load status of each group as well. By monitoring the product line data as shown in Table V, planned and completed work follow-up can be carried out. Using the statistical data gathered, resource planning and productivity performance analysis for each software module & team can be successfully accomplished. By taking goals and productivity coefficients into account, pre-detection actions for restructuring the problematic software modules can be put into practice in the future.

F. Cost and Planning Process Observations

In order to count functional size of any request, functional requirements are needed. In the beginning of the transition phase, it was hard to complete the cost estimation process because of lacking required information regarding the request requirement specifications. Therefore, to determine functional and non-functional requirements for estimating approximate costs for requests, meetings with the participation of module owners and FP practitioners are being held.

After calculating approximate costs of the requests, the requests are planned by putting them on the production line using available resources. Then as shown in Figure 4 and 5, analysis, high level design, and design steps are performed before the requests reach the final cost estimation step. On this step, final cost is reckoned using the analysis and design documents. Once the final cost estimation is complete and approved by PO, software development efforts may begin using the available resources.

 ISBN: 978-1-61208-367-4

TABLE V. THE MAIN CONTRACTOR'S PRODUCTION LINE PLANNING AND PRODUCTION TRACKING.

PLANNED		2012	January	February	March	April	May	June	July	Total
TB-II	Core Banking II	1.168	347	751	447	674	476	625	0	4.487,33
TB-I	Core Banking I	961	464	532	991	636	775	630	0	4.988,80
BA	Banking Infrastructure	209	252	390	613	513	409	353	0	2.738,66
DU	Support Systems	423	260	278	277	236	135	216	0	1.825,49
İZKD	Business Intelligence Support System	352	186	261	388	138	788	32	0	2.143,88
TB-III	Core Banking III	427	26	86	540	250	49	42	0	1.419,49
ÜY	Product Management	933	0	0	0	37	0	0	0	969,80
VY	Data Management	0	290	72	0	13	0	0	0	375,63
İZMİR	İzmir	10	0	0	0	220	7	0	0	236,06
TOTAL		4484	1824	2369	3256	2716	2638	1897	0	19.185,13

COMPLETED		2012	January	February	March	April	May	June	July	Total	Goal	%
TB-II	Core Banking II	113	346	592	375	524	141	100	0	2.189,24	7.100,00	30,83%
TB-I	Core Banking I	531	403	482	394	321	230	28	0	2.388,66	7.050,00	33,88%
BA	Banking Infrastructure	163	252	371	286	371	156	76	0	1.676,09	3.910,00	42,87%
DU	Support Systems	142	247	199	175	70	49	3	0	883,99	2.415,00	36,60%
İZKD	Business Intelligence Support System	34	159	217	110	116	95	6	0	736,89	1.945,00	37,89%
TB-III	Core Banking III	18	26	18	131	31	7	4	0	235,48	1.850,00	12,73%
ÜY	Product Management	0	0	0	0	0	0	0	0	0,00	909,00	0,00%
VY	Data Management	0	0	0	0	2	0	0	0	2,52	180,00	1,40%
İZMİR	İzmir	4	0	0	0	44	3	0	0	51,73	150,00	34,49%
TOTAL		1006	1432	1879	1472	1479	681	216	0	8.164,60	25.509,00	32,01%

As can be seen in Table V, in the new cost system, FP calculations in the transition period are less than the other periods. Total FP for January 2013 is 1824 and more requests are being loaded to the production line in the following months. The reasons behind the weak performance in the transition period are technical problems, personnel resistance fed from old habits and efforts spent for obtaining functional requirements. In the succeeding step of our process enhancement, one to one negotiations for functional requirement inference, which is examined in the transition period, are left and a new document, namely preliminary requirement analysis document, is organized with the contributions of business analysts and module owners in order to calculate true approximate cost.

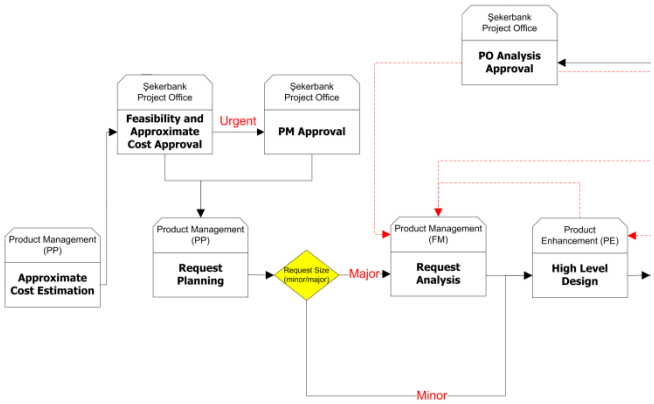


Figure 4. Software Production Process Pipeline Part 1.

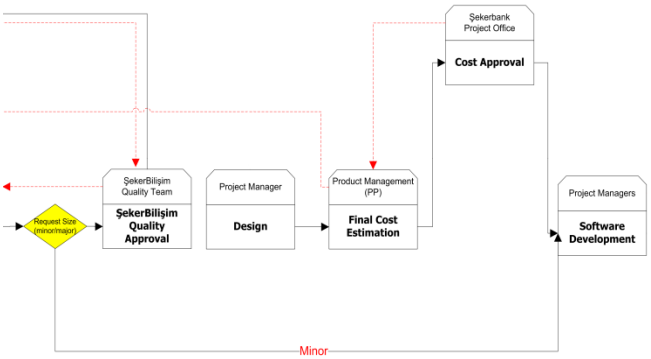


Figure 5. Software Production Process Pipeline Part 2.

G. Scoping Meetings

One of the ongoing improvement efforts is improving the efficiency of scoping meetings, which are performed at an intense pace. In order to perform more effective and more conscious monthly and quarterly plans, comprehensive requirement gathering activities are conducted for creation of a request pool, which consists of requests that have initial cost estimations. Project office, business unit, business analysts, module owners and production planning experts are participating in these activities. Utilizing the outcomes of the scoping meetings, due to assessing the situation of the production line from a wider perspective, long-term business targets will be identified and prioritized.

1) Observations: Difficulties in requirement gathering activities, especially requests that require integration of different modules, are noted and it is anticipated to cause inconveniences for accurate cost estimation efforts. However, in order to increase the efficiency of the software development efforts, we desire to minimize the participation of the relevant module's software engineers in requirement gathering activities. However, considering the lack of technical background of the business analysts at the moment, software engineers are still important assets for the scope meetings.

VI. FUTURE WORK

Because of historical reasons, software engineers have led scoping and requirements gathering activities. This role actually belongs to business analysts and we need to train them to increase their competence in business architecture. Accordingly, business analysts will contribute more in the scoping and requirements gathering meetings, so this affects efficiency of the Software Production Line, since software engineers will involve less in these meetings and activities, and focus only software development phase. Moreover, once problematic modules will be identified by observing productivity ratios, Product Improvement team will conduct necessary restructuring and re-factoring activities.

When it comes to cost estimation process, it is under continuous quality control, which let us fine-tune of productivity calculations. Establishing Software Product Line will be much easier after measuring all metrics of software production line, which is the next goal of the company.

VII. CONCLUSION

The main contractor and the main supplier have been capable of measuring several metrics related to the software production line via IFPUG functional size measurement

 ISBN: 978-1-61208-367-4

method. In the process of adaptation to the new system, a resistance fed from old habits is faced and new steps have been added to the production process to achieve the desired effect in requirement gathering activities.

Within the scope of adaptation of function points to the production line, arrangements are made on existing request types, request sizes, and project types. In order to estimate total cost for different request types with different request sizes on various project types, process factor is defined and is used as shown in the Table III. Simple COCOMO equations are adapted for FP and statistical data of the product line, hereby, are gathered. In consequence of available data, the condition of the production line can show the actual and planned works along with the accumulated workload on the business units. Making use of these indicators, production and resource planning can be made more efficiently and factors adversely affecting the process can be observed.

Scoping meetings are made in requirement of detailed information to accurately estimate cost of a request and new methods for the solution are actively being searched. Annual software development goals can be determined by productivity calculations that are based on FP for each team. By utilizing productivity factors, modules that have low productivity performance are identified. Once the identification process is complete, the identified modules are targeted for restructuring purposes to improve development productivity.

REFERENCES

[1] C. Jones, Applied software measurement: global analysis of productivity and quality. McGraw-Hill, 2008, ISBN 978-0-07-150244-3.

[2] M. Solla, A. Patel, C. Wills, "New metric for measuring programmer productivity," Proc. IEEE Symp. Computers and Informatics, IEEE, 2011, pp. 177-182.

[3] N. Ramasubbu , M. Cataldo , R. K. Balan , J. D. Herbsleb, "Configuring global software teams: a multi-company analysis of project productivity, quality, and profits," Proceedings of the 33rd International Conference on Software Engineering, USA, May 21-28, 2011, pp. 261-270.

[4] O. T. Pusatli, S. Misra, "Software Measurement Activities in Small and Medium Enterprises: an Empirical Assessment," Acta Polytechnica Hungarica, 8 (5) , 2011, pp. 21-42.

[5] J. S. Poulin, "The Economics of Software Product Lines," International Journal of Applied Software Technology 3, 1997, pp. 20–34.

[6] P. Clements, J. McGregor, S. Cohen, The Structured Intuitive Model for Product Line Economics (SIMPLE) (CMU/SEI-2005-TR-003). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.

[7] S. B. Lamine, L. L. Jilani, H. H. B. Ghezala, "A Software Cost Estimation Model for a Product Line Engineering Approach: Supporting tool and UML Modeling," 3rd ACIS Conf. on Software Engineering Research, Management and Applications, 2005, pp. 383-390.

[8] J. P. Nóbrega, E. S. Almeida, S. R. L. Meira, "InCoME: Integrated Cost Model for Product Line Engineering," Proceedings of the 2008 34th Euromicro Conference Software Engineering and Advanced Applications, 2008, pp. 27-34.

[9] A. J. Nolan, S. Abrahão, "Dealing with Cost Estimation in Software Product Lines: Experiences and Future Directions," SPLC'10 Proceedings of the 14th international conference on Software product lines, 2010, pp. 121-135.

[10] N. I. Altintas, M. Surav, O. Keskin, and S. Cetin, "Aurora software product line," 2nd National Software Engineering Conference, Ankara, Turkey, Sep. 2005.

[11] N. I. Altintas, S. Cetin, A. H. Dogru, and H. Oguztuzun, "Modeling Product Line Software Assets Using Domain-Specific Kits," IEEE transactions on software engineering, vol. 38, Dec. 2012, pp. 1376-1402.

[12] D. E. Nye, America's Assembly Line. MIT Press, 2013, ISBN 978-0262018715.

[13] P. R. Hill, Practical software project estimation: a toolkit for estimating software development effort and duration. McGraw-Hill, 2010, ISBN 978-0-07-171791-5.

[14] A. J. Albrecht, "Measuring application development productivity," IBM Application Development Symposium, OCT. 1979, pp. 83-92.

[15] ISO/IEC 19761, 2003 COSMIC Method Measurement Manual v. 3.0.1.

[16] ISO/IEC 29881, 2008 Information technology Software and systems engineering FiSMA 1.1 functional size measurement method.

[17] ISO/IEC 20926, 2009 Software Engineering - IFPUG 4.3.1. Unadjusted FSM Method - Counting Practices Manual.

[18] ISO/IEC 20968, 2002 Software Engineering - Mk II Function Point Analysis - Counting Practices Manual.

[19] ISO/IEC 24570, 2005 Software Engineering - NESMA Functional Size Measurement Method v.2.1 - Definitions and counting guidelines for the application of Function Point Analysis.

[20] Automated Function Points (AFP) Version 1.0 - Beta 1, ptc/2013-02-01.

[21] The international function point users group: function point counting practices manual release 4.3.1., 2010, ISBN 978-0-9753783-4-2.

[22] Borland Conference, How to determine your application size using function points. [Online]. Available from http://conferences.embarcadero.com/article/32094.

[23] P. Morris, "Mapping the rules for IFPUG and COSMIC-FFP function size method," IFPUG Fall Conference, Scottsdale, Arizona, USA, 2003, pp. 299-319.

[24] Common Software Measurement International Consortium, A Comparison of the Key Differences between the IFPUG and COSMIC Functional Size Measurement Methods. [Online]. Available from: http://www.cosmicon.com/portal/public/IFPUG_COSMIC_Key_Comparison.pdf.

[25] B. W. Boehm, Software engineering economics. Englewood Cliffs, NJ:Prentice-Hall, 1981, ISBN 0-13-822122-7.

 ISBN: 978-1-61208-367-4

Empirical Research in Software Engineering: A Literature Review

Petr Pícha, Přemysl Brada
Department of Computer Science and Engineering
University of West Bohemia
Pilsen, Czech Republic
{ppicha, brada}@kiv.zcu.cz

***Abstract*-Software engineering (SE) and the empirical research in this area have both become a large fields with significant bodies of knowledge as well as methodical problems. Mentions, descriptions and examples of these problems are spread throughout the literature, but only a handful of suggestions and attempts of their rectification are being presented. This paper summarizes the goals and history of the SE field and focuses on the empirical research area within it. It highlights the most frequent problems affecting empirical SE research efforts and their most promising suggested solutions. Both the problems and suggested solutions were collected from a carefully selected sample of research publications. The presented overview should serve as a starting point for researchers or other professionals trying to get the first broad and shallow insight into the context of SE empirical research, as well as a theoretical basis for subsequent research.**

Keywords-empirical research, literature review, software engineering, history, problems, solutions

I. INTRODUCTION

The field of Software Engineering (SE) has been developing since the 1960s, and has, therefore, amassed a substantial amount of knowledge. Even empirical SE research, which specializes in studies based on experiments and observation, is very rich and obtaining the full grasp of its topics and findings poses a significant challenge.

The goal of this work was to perform a broad and shallow study of literature concerning the current state of Empirical Research in Software Engineering (ERSE). The motivation for this study was to obtain an initial understanding of the area as a prerequisite for further, deeper research on its particularities, details and various research methods and approaches.

The challenge of the study was obviously the extent of the literature body obtained. Therefore, this paper focuses on summarizing the findings about ERSE and the SE discipline in general. It highlights their context, goals, major problems in conducting empirical SE research and some suggested solutions.

The study will be useful mainly for researchers starting in the SE research field trying to gain a general overview of its content and context quickly and with minimal effort.

In section II, we briefly describe the motivation for and design of the study. Section III speaks about similar papers included in the studied literature sample and the main differences between this them and this paper. Section IV presents facts and findings about ERSE and SE in terms of their goals and history. Section V describes the major problems of the field found in literature while Section VI mentions the proposed solutions to these problems. Finally, Section VII discusses the accuracy and relevance of this study. Section VIII summarizes the principal findings and explains their proposed usefulness for other researchers and other readers.

II. STUDY CONTEXT AND DESIGN

This section explains the main reasons for creating this literature review and its goals as well as a process of its execution.

A. Reasons for the Study

The main motivation for this study was to create a pivotal resource and knowledge basis for ERSE within the local research group (Reliable Software Architecture - ReliSA) at the University of West Bohemia. To establish a baseline before any particular studies could be conducted we needed to obtain an overview of the ERSE field itself in a broad, but not necessarily too detailed manner.

In particular, we wanted to find out key information about the following aspects of ERSE: its context; methodologies and taxonomies used; leading experts, organizations and outlets; major SE problems studied in the research community and pitfalls of the research approaches both in general and in their particular steps (including ways to avoid or mitigate them). The literature review fitted all these goals as the best way to achieve them.

B. Process of Study Execution

The design of the study (meaning the process of literature gathering and exploration in order to extract the key information sought) breaks down to several steps.

At first, the main sources of material were selected. These were the academically oriented part of the search engine Google - Google Scholar, and digital libraries (DL) of scientific publications, such as ACM DL, IEEE DL and Springer Link. The key expressions, such as “empirical research”, “software engineering experiment”, or “research in software engineering” were searched in these sources to get a preliminary set of publications and organizations.

 ISBN: 978-1-61208-367-4

Then, the websites of the organizations that occurred most frequently and seemed most promising were explored to widen the set of publications. The relevance of the publications was determined mainly by reading their abstracts.

The whole set of selected publications was sorted using a simple citation metric which takes the sum of the publication's citation counts on Google Scholar and Web Of Science and divides this number by the age of the publication in years plus one (so that the papers published in the current year do not end up with count divided by zero).

The next step was the actual reading of the materials. Mainly, only the papers were read; the books were omitted for reasons of the exceedingly long time required to read them.

After reading each paper, the referenced publications were checked for relevance and, if found useful, added to the set of useful publications with their citation metric value calculated.

Whenever a significantly large subset of newly added papers originated from one organization previously unexplored, the website of the organization was checked and relevant papers were added as before.

The whole process stopped at the point where we had a list of the top six reachable papers which remained unchanged after studying all relevant referenced publications, i.e., when the process stopped uncovering new significant papers.

C. The Analyzed Literature Sample

The end result of the process described in the previous subsection was a set of almost 90 collected publications, with a subset of 53 publications found both available and useful. The publication years of this subset range from 1986 to 2014.

In terms of outlet, in which these publications were published, most of them (approximately 72%) came from journals, magazines, newsletters and other periodic publications. Most significant contributions amongst them made Information and Software Technology (Elsevier), Transactions on Software Engineering (IEEE), Empirical Software Engineering (Springer), and Software (IEEE). Almost 21% of all the publication studies came from conferences, symposiums and meetings with International Conference on Software Engineering (ICSE), International Symposium on Empirical Software Engineering and Measurement (ESEM) and The Future of Software Engineering Symposium (FOSE) leading the list. The rest of the publications (roughly 8%) originated elsewhere, e.g., were a technical report or an available chapter of a book.

On the level of the organizations from where the publications originated in the majority of cases (approximately 70%) at least one author was affiliated with Simula Research Laboratory (Lysaker, Norway), Keele University (UK), University of Maryland (USA), NICTA (National Information Communications Technology Australia – University of New South Wales, Sydney), Lund University (Sweden), or SINTEF (The Foundation for Scientific and Industrial Research, Trondheim, Norway). Other organizations included Microsoft's ESE (Empirical Software Engineering Group), Karlsruhe Institute of Technology (Germany), Chalmers University of Technology (Gothenburg, Sweden), Carnegie Mellon University (Pittsburgh, USA) and other institutions from USA, Ireland, Canada, Norway, UK and France.

III. Related Work and Similar Studies

This study certainly does not constitute the first attempt in the research community to assess the state and context of the SE field and ERSE area. This section mentions the similarly oriented studies closest to ours. It also highlights the differences and individual properties of our paper in relation to them.

Glass et al. [1] discuss the history of SE research and then try to categorize 369 selected papers from 1995 to 1999 on several levels. Perry et al. [2] try to assess the contemporary state and future (at the time) challenges of empirical studies in SE followed by describing the recommended structure of an individual study and concrete steps for assuring its quality. Sjöberg et al. [3] give the description of primary and secondary research methods followed by a vision for increased research quality for the future and suggestions leading to its fulfilling. Victor Basili published his study overview of the field in [4] including the nature of the discipline, research paradigms, vision and its attainability, goals of studies, types of studies and a description of a maturing process of the research.

None of the mentioned (or other studied) publications deals strictly with summarizing the general context of SE and ERSE in terms of their goals, history, major problems and solutions. The main contribution of this particular paper lies in summarizing the problems and solutions on the level of the field and individual studies (not the concrete phases and steps while conducting research). In addition, the literature sample used in our study spreads through almost three decades. That makes its findings not only contemporary, but also a broad overview, since none of the above mentioned works describes all the problems and solutions included in this paper.

IV. Context of Software Engineering Research

This section presents findings about SE, SE research and ERSE in terms of goals and history discovered throughout the studied literature.

A. Goals of Research

As in other scientific disciplines, the main goal of SE research is to contribute to the effectivity and provide knowledge for decision making in its respective field both in further research and in practice. However, the outcomes of the research can be used by several different types of audiences, such as common readers, other researchers, reviewers, meta-analytics, different kinds of committees and practitioners. While the resulting papers should ideally bring some level of benefit to them all, the authors may have to decide beforehand who is their target audience and how the

 ISBN: 978-1-61208-367-4

study should benefit them and adjust the study and its goals properly.

Of course, there is the matter of subsequent research. The occurrence of one paper which brings great benefit to the field as a whole on some 10 pages is hardly imaginable. So the research in particular organizations (or several cooperating organizations) should be carefully designed and structured to support the preliminary vision and high-level goals.

Much has been written about how the individual studies should be done and research papers structured [2][5][6][7][8][9][10]. Most of the findings in this area will be summed up in a future work; here were we would like to present just one of the more general ideas suggested in [3], which claims that the point of every study is (or at least should be) the exploration or description of the relations between four "archetype classes". These classes are: Actor, Technology, Activity and Software System. In this context, Actor can be not only a person or a group of people but also another SW or HW system and so on. Furthermore, Technology class representative need not to be a piece of HW or SW exclusively. It can also refer to a technique, method, practice, diagram, model, guideline, etc. This seems to be a pretty straightforward, simple and achievable principle.

B. Software engineering history

Now, let us talk a bit about SE and ERSE history. The most useful, yet general and complex description of history was found in [1], which is the main source of the information presented here.

The existence of SE can be traced back to the early 50s. The SE research (including ERSE) appeared almost simultaneously, although it was mainly ad hoc research and unfortunately, though much has been researched and discovered, there were very few outlets to present the findings in. This lead to the fact, that many of the findings from this era were published roughly 10 years afterwards.

And so the real traceable history of SE research dates back to the late 60s when the first SE conferences were being held. However, back in that time the SE was still just one part of the much larger Computer Science field, and did not begin to separate its presence in academia off from it until the early 80s.

Of course, that was not the end of the genesis of the field that continues up until this day. For example, qualitative research (e.g., [11]), which is aimed more on how things work (especially important in SE, since the impact of the human factor on the field is very significant) than on causal links and numbers, did not really appear in noticeable measure until 10 to 20 years ago and still does not have very large volume compared to standard and more "technical" quantitative research. (There is a simple and useful differentiation between qualitative and quantitative research methods in [7].)

V. Problems in Software Engineering Research

The studied literature identifies and describes many problems of SE research impacting both the field in general (presented in this section), or just one or more steps in particular studies (described in a future work). Many of these problems appear in more than one publication and this section summarizes and presents the ones that seem the most significant and frequently encountered.

A. The Whole Field

At the top of the list is probably the **absence of** any **unified** and universally recognized **guidelines, methodologies, taxonomy and** even **terminology** which could be used and followed while conducting studies and presenting the results in research papers. Some authors [2][5][6][7][12][13][14] try to come up with first suggestions or their own models for research or its evaluation, but no field-wide agreement has been reached so far. The result of this is an inconsistency of studies and papers throughout the research community, which leads to sort of isolation of the studies, that cannot be compared, widened, summarized by meta-analysis or theory building, followed up by other studies, or even properly disproved.

A related problem is often mentioned **little support in recognized authority** which could come up with these so needed guidelines and methodologies, or at least support their development and mediate a discussion on them, or even make them a standard in the end. Examples from other fields given in literature include the *Cochrane Collaboration* and their *Cochrane Database of Systematic Reviews* (Australia; mentioned in [15]) in medicine, or the *Human Genome Project* (mentioned in [3]) in genetics. Some sort of comfort or hope lies in *SWEBOK* which describes issues related to SE but still does not include guidelines for research.

Another problem is that **not enough research** in general. According to some (e.g., [3], [16]), too few studies are being conducted, presented and published in the SE and ERSE field. This is of course a matter of opinion, but the truth is the more studies undertaken the better. Even if their quality (further discussed later in this paper) is in general poor, the total number of good papers is higher in bigger overall amount than in smaller. There are some subareas where the current situation is even worse.

One is **qualitative research** (already mentioned in Section IV), which **is** somewhat **rare** in itself in the SE field. Although the situation started to get better in recent years, as finally more and more qualitative research (described for example in [11] or [17]) is done, the pace is not fast enough.

The other is **theory building** [3][18][19][20][21], which **suffers** from subsequent issues from incorrect conclusions (mentioned in [5]), ignoring negative results (mentioned in [22]), research questions not being insightful enough (mentioned in [2]), misused statistical methods (described, e.g., in [23], [24]) and poor quality of studies in general (many of these problems are mentioned in following subsection B) which, of course, makes it difficult to prove hypotheses and build theories, which are necessary for establishing some ground on which to build further activities in the field.

 ISBN: 978-1-61208-367-4

The next problem often discussed and mentioned is the **gap between practice** of the field **and academia** (or the research community in general). This results in research not covering the real needs of the industry and in return not applying the findings of the research in practice. This has an immediate link to the next issue (next paragraph), but another statement found in [1] linked to this talks about so-called "**assimilation gap**", which refers to the time frame in between the first acquisition of a new technology (tool, technique, method, practice, model, etc.) and its 25% penetration into software development organizations in practice, and according to [1], is nowadays 9 to 15 years wide.

A large amount of **research is purposeless** as mentioned, e.g., in [2]), meaning its being done simply for something to be done without the aims and goals of the research corresponding to or reflecting on the needs of the practice. The cause of this, apart from the above mentioned gap between industry and academia, is the fact that often a major evaluation criterion of researchers (including the PhD students) is the quantity of published papers and citations. And so, if the researchers have no experience or relationship with the actual practice in the respective field, they end up doing research just for its own sake.

The last big issue of the field as a whole is the **lack of resources**, both financial and personal. The ERSE still struggles to get sufficient support from governments, industry or other entities, such as different kinds of initiatives and foundations. The comparison with other fields, especially medicine, is often given in the literature studied [3].

This concludes the problems of the field in general. Other listed problems refer more or less to the issues that the individual studies and papers suffer from.

B. Individual Studies

The main problem concerning individual studies is **low quality** (see, e.g., [25])**, relevance** (meaning significance of results) **and usability** (also called *impact* – factor of research being interesting for the field; [8]). Many studies suffer from **misusage of statistical methods**, which are often poorly understood by IT practitioners [23]. They also often just present results or state conclusions with **improper or no validation and evaluation** whatsoever, as mentioned in [8][12][15][16][26][27].

Another contributing factor is **incorrect conclusions drawing** (see, e.g., [5])**, fishing for results** (presenting insignificant results so that there is something to publish) and mostly **ignoring negative results** (as mentioned, e.g., in [22]). Although not positive (in terms of proving the hypothesis or showing something is wrong with the studied phenomenon), such results still need to be published or made public at the very least to prevent other researchers from repeating the same thing once it has been proven meaningless. Furthermore, these results can still be followed by validation studies, replication studies (see, e.g., [28]), qualitative studies shedding light on what went wrong, etc.

The follow-up studies are often difficult to conduct for one more reason and that is **publishing** study results **without the** input **data** used. This, of course, has its source in the fact that the data are mostly industrial by origin and as such considered sensitive and in need of keeping them confidential, which is unfortunately something SE researchers can hardly influence or overcome (as mentioned, e.g., in [29]).

Last but not least, in reference to the above mentioned gap between industry and academia (subsection A), the researchers are often given no choice but to use **students as subjects in** their **studies**. The reasons are obvious: students are easily available and their usage is cheap [9][22][30]. But, of course, practitioners tend to dismiss the results of such studies as irrelevant or not representative of reality instead of considering the results and if found interesting trying to supply replication study with their own people.

VI. SUGGESTED SOLUTIONS TO THE PROBLEMS

This section deals with solutions to the problems in Section V, suggested in the studied literature.

As many publications compare the state of SE and ERSE to other disciplines with longer history and more stable infrastructure, the first suggested course of action is **patterning** our field **after,** such **other disciplines** (e.g., [26][31]) like medicine, psychology and sociology (both for their overlap with IT in SE and long tradition of qualitative research), information systems, or computer science. This could potentially bring some "order" to the "chaos" and establish a platform for a developing more field-specific empirical research framework.

Another suggestion was to **include practitioners and statisticians** into the research activities (as mentioned in [5][9]). This would significantly improve the situation concerning the gap between academia and industry, purposeless research, misuse of statistical methods, validation and evaluation of the results. The problem here is of course the above mentioned resources issue.

The resources could be brought by initiatives and organizations (see, e.g., [9]) established for these purposes and also for the purposes of stabilizing the terminology, taxonomy, methodology and guidelines, as well as theory building and authority foundation. Of course, to **establish** such **authorities and initiatives**, community-wide cooperation towards these goals is essential, but once done this should have major impact on research quality and effectivity, thus fulfilling the purpose of research in the first place (section IV). A way to collaborate on the community level through social networking is described in [32].

But before that can be done, the lower-level cooperation needs to appear and grow. Research organizations such as universities, institutes and even companies should strive to **build relationships** on a common goal of research and improvement [2][3]. Furthermore, each research entity should have some degree of **long-term** focus on a particular **topic** (or topics, depending on size and manpower), and conduct not isolated studies, but whole **families of studies** surrounding the topic [2][28]. The similar topic focus among several entities should supply the common ground for the above mentioned cooperation.

 ISBN: 978-1-61208-367-4

In terms of individual studies one more issue has to be mentioned and that is the **insightfulness of** research **questions** (see, e.g., [2]). The importance of this is that if you do not have research questions or hypotheses insightful enough, you undermine your studies from the very beginning. And although you may still finish the studies and present your results, such studies are hard to follow up on, replicate or include in meta-analysis (see, e.g., [33][34]), rendering them isolated, and therefore, useless.

The last main suggested solution looks to the future in a different way. Instead of trying to change the field, current researchers, practitioners and their mindset, the solution lies in the upbringing of good researchers from the very beginning – of course through **education**. The literature [2][35] suggested special SE research courses at colleges and universities to try to prepare students with research aspirations for the obstacles and circumstances of such a career and to teach them the basic knowledge on how to do research properly. This would, however, better be premeditated by establishing proper, unified and universally acknowledged methodologies and guidelines (mentioned before), or else every education entity will end up teaching their students something different and the resulting chaos will be little to none better than the current one.

VII. DISCUSSION

Here, we discuss the usage and relevance of this work. As it is a simple literature review we do not claim that the information presented is fully comprehensive and accurate. Much wider and deeper analysis would be needed for a statistically valid study.

Through the course of this paper, we often talk about SE research, while the topic should be mainly ERSE. This is caused by the broad and shallow nature of this study. In its high-level context, the separation of empirical research off from SE research in general is difficult and probably not feasible to some degree. Nevertheless, the findings apply to both areas in most cases and where not, it is impossible to describe the issues of ERSE without the wider context of SE research.

Lastly, this presented paper does not cover the full breadth of the studied material and findings. As future work, we plan to address the methodology and taxonomy aspects of ERSE, various types of studies, suggested steps in conducting and describing the studies in papers and of course basic techniques, principles, pitfalls, and frequent mistakes made in each of these steps.

VIII. CONCLUSIONS

This literature review summarized and described information and findings discovered throughout the studied publications concerning SE and ERSE in this field.

The studied literature sample consisted of 53 papers and articles published from 1986 to 2014, coming from major journals and conferences in the field and spread throughout the world in their origin. The key problems in the area of SE research, as found through this study, are overall insufficient quality, relevance and impact. The steps towards mitigation and avoidance of these problems are being taken throughout the respective community, although it is a long-term and slow process.

The general advice for improving the research quality includes rigorous study design and description (correct usage of statistical methods, drawing of conclusions, insightful research question, validation and evaluation), including statisticians and as much of practitioners participation (or at least industrial data) to the research as possible, education of researchers through specialized courses, establishing authorities and financial support systems, and using other scientific disciplines as a pattern.

Our hope is that this review helps especially other scientists starting in the ERSE field as a base of general knowledge and overall overview. In addition, the findings can be used as an information basis for both further and more detailed research and conducting individual empirical studies in the field of software engineering.

ACKNOWLEDGMENT

The work was supported by the UWB grant SGS-2013-029 Advanced Computer and Information Systems.

REFERENCES

[1] R. L. Glass, I. Vessey, and V. Ramesh, “Research in Software Engineering: An Analysis of the Literature”, Information and Software Technology, vol. 44, issue 8, Jun. 2002, pp. 491-506, doi: 10.1016/S0950-5849(02)00049-6.

[2] D. E. Perry, A. A. Porter, and L. G. Votta, “Empirical Studies of Software Engineering: A Roadmap”, Proceedings of the Conference on The Future of Software Engineering (ICSE ‘00), ACM New York, Jun. 2000, pp. 345-355, ISBN: 1-58113-253-0, doi: 10.1145/336512.336586.

[3] D. I. K. Sjöberg, T. Dyba, and M. Jörgensen, “The Future of Empirical Methods in Software Engineering Research”, Future of Software Engineering (FOSE ‘07), IEEE, May 2007, pp. 358-378, ISBN: 0-7695-2829-5, doi: 10.1109/FOSE.2007.30.

[4] V. R. Basili, “The Role of Experimentation in Software Engineering: Past, Current and Future”, 18th International Conference on Software Engineering (ICSE ‘96), IEEE Computer Society, 1996, pp. 442-449, ISBN: 0-8186-7246-3.

[5] B. A. Kitchenham, S. L. Pfleeger, D. C. Hoaglin, K. E. Eman, and J. Rosenberg, “Preliminary Guidelines for Empirical Research in Software Engineering”, IEEE Transactions on Software Engineering, vol. 28, issue 8, Aug. 2002, pp. 721-734, doi: 10.1109/TSE.2002.1027796.

[6] M. Shaw, “What Makes Good Research in Software Engineering?”, International Journal on Software Tool For Technology Transfer, vol. 4, issue 1, pp. 1-7, Oct. 2002, doi: 10.1007/s10009-002-0083-4.

[7] C. Wohlin, M. Höst, and K. Henningsson, “Empirical Research Methods in Software Engineering”, Empirical Methods and Studies in Software Engineering, Springer Berlin Heidelberg, pp. 7-23, 2003, ISBN: 978-3-540-40672-3, doi: 10.1007/987-3-540-45143-3_2.

[8] T. Dyba, B. A. Kitchenham, and M. Jörgensen, “Evidence-based Software Engineering for Practitioners”, IEEE Software, vol. 22, issue 1, Jan./Feb. 2005, pp. 58-65, doi: 10.1109/MS.2005.6.

[9] S. S. Briliant and J. C. Knight, “Empirical Research in Software Engineering: A Workshop”, ACM SIGSOFT Software Engineering Notes (Newsletter), vol. 24, issue 3, May 1999, pp. 44-52, doi: 10.1145/311963.311998.

[10] B. A. Kitchenham, ”Procedures for Performing Systematic Reviews”, Joint Technical Report, July, 2004, ISSN: 1353-7776.

 ISBN: 978-1-61208-367-4

[11] C. B. Seaman, "Qualitative Methods in Empirical Studies of Software Engineering", IEEE Transactions on Software Engineering, vol. 25, issue 4, Jul./Aug. 1999, pp. 557-572, doi: 10.1109/32.799955.
[12] V. R. Basili and R. W. Selby, D. H. Hutchens, "Experimentation in software engineering", IEEE Transactions on Software Engineering, vol. 12, issue 7, Jul. 1986, pp. 733-743, doi: 10.1109/TSE.1986.6312975.
[13] B. A. Kitchenham et al., "Can We Evaluate the Quality of Software Engineering Experiment?", The Forth International Symposium on Empirical Software Engineering and Measurement (ESEM '10), ACM New York, Sep. 2010, article no. 2, ISBN: 978-1-4503-0039-1, doi: 10.1145/1852786.1852789.
[14] M. Höst and P. Runeson, "Checklists for Software Engineering Case Study Research", The First International Symposium on Empirical Software Engineering and Measurement (ESEM '07), IEEE, Sep. 2007, pp. 479-481, ISBN: 978-0-7695-2886-1, doi: 10.1109/ESEM.2007.46.
[15] B. A. Kitchenham, T. Dyba, and M. Jörgensen, "Evidence-based Software Engineering", The 26th International Conference on Software Engineering (ICSE '04), IEEE, May 2004, pp. 273-281, ISBN: 0-7695-2163-0, doi: 10.1109/ICSE.2004.1317449.
[16] W. F. Tichy, "Should Computer Scientists Experiment More?", IEEE Computer, vol. 31, 1997, pp. 32-40.
[17] M. Dixon-Woods, "How Can Systematic Reviews Incorporate Qualitative Research? A Critical Perspective", Qualitative Research, vol. 6, issue 1, Feb. 2006, pp. 27-44, doi: 10.1177/1468794106058867.
[18] M. Jörgensen and D. Sjöberg, "Generalization and Theory-Building in Software Engineering Research", The 26th International Conference on Software Engineering (ICSE '04) - Workshop "8th International Conference on Empirical Assessment in Software Engineering (EASE 2004)", May 2004, pp. 29-36, ISBN: 0-86341-435-4, doi: 10.1049/ic:20040396.
[19] A. Zendler, "A Preliminary Software Engineering Theory as Investigated by Published Experiments", Empirical Software Engineering, vol. 6, issue 2, Jun. 2001, pp. 161-180, doi: 10.1023/A:1011489321999.
[20] D. I. K. Sjöberg, T. Dyba, B. C. D. Anda, and J. E. Hannay, "Building Theories in Software Engineering", Guide to Advanced Empirical Software Engineering, Springer London, pp. 312-336, 2008, ISBN: 978-1-84800-043-8, doi: 10.1007/978-1-84800-044-5_12.
[21] J. E. Hannay, D. I. K. Sjöberg, and T. Dyba, "A Systematic Review of Theory Use in Software Engineering Experiments", IEEE Transactions on Software Engineering, vol. 33, issue 2, Feb. 2007, pp. 87-107, doi: 10.1109/TSE.2007.12.
[22] W. F. Tichy, "Hints for Reviewing Empirical Work in Software Engineering", Empirical Software Engineering, vol. 5, issue 4, Dec. 2000, pp. 309-312, doi: 10.1023/A:1009844119158.
[23] T. Dyba, V. B. Kampenes, and D. I. K. Sjöberg, "A Systematic Review of Statistical Power in Software Engineering Experiments", Information and Software Technology, vol. 48, issue 8, Aug. 2006, pp. 745-755, doi: 10.1016/j.infsof.2005.08.009.
[24] B. A. Kitchenham, D. R. Jefferey, and C. Connaughton, "Misleading Metrics and Unsound Analyses", IEEE Software, vol. 24, issue 2, Mar./Apr. 2007, pp. 73-78, doi: 10.1109/MS.2007.49.7.
[25] M. Jörgensen, T. Dyba, K. Liestöl, and D. I. K. Sjöberg, "Incorrect Results in Software Engineering Experiments: How to Improve Research Practices", unpublished.
[26] M. V. Zelkowitz and D. Wallace, "Experimental Validation in Software Engineering", Information and Software Technology, vol. 39, issue 11, 1997, pp. 735-743, doi: 10.1016/S0950-5849(97)00025-6.
[27] B. A. Kitchenham et al., "Evaluating guidelines for reporting empirical software engineering studies", Empirical Software Engineering, vol. 13, issue 1, Feb. 2008, pp. 97-121, doi: 10.1007/s10664-007-9053-5.
[28] V. R. Basili, F. Shull, and F. Lanubile, "Building Knowledge through Families of Experiments", IEEE Transactions on Software Engineering, vol. 25, issue 4, Jul./Aug. 1999, pp. 456-473, doi: 10.1109/32.799939.
[29] V. R. Basili, M. V. Zelkowitz, D. I. K. Sjöberg, P. Johnson, and A. J. Cowling, "Protocols in the Use of Empirical Software Engineering Artifacts", Empirical Software Engineering, vol. 12, issue 1, Feb. 2007, pp. 107-119, doi: 10.1007/s10664-006-9030-4.
[30] D. I. K. Sjöberg, B. Anda, and T. Dyba, "Conducting Realistic Experiments in Software Engineering", 2002 International Symposium on Empirical Software Engineering, IEEE, Oct. 2002, pp. 17-26, ISBN: 0-7695-1796-X, doi: 10.1109/ISESE.2002.1166921.
[31] D. Budgen et al., "Cross-domain Investigation of Empirical Practices", IET Software, vol. 3, issue 5, Oct. 2009, pp. 410-421, doi: 10.1049/iet-sen.2008.0106.
[32] A. Begel, J. Bosch, and M. A. Storey, "Bridging Software Communities through Social Networking", IEEE Software, vol. 30, issue 1, Jan. 2013, pp. 26-28, doi: 10.1109/MS.2013.3.
[33] J. Miller, "Applying Meta-analytical Procedures to Software Engineering Experiments", Journal of Systems and Software, vol. 54, issue 1, Sep. 2000, pp. 29-39, doi: 10.1016/S0164-1212(00)00024-8.
[34] L. M. Pickard, B. A. Kitchenham, and P. W. Jones, "Combining Empirical Results in Software Engineering", Information and Software Technology, vol. 40, issue 14, Dec. 1998, pp. 811-821, doi: 10.1016/S0950-5849(98)00101-3.
[35] M. Jörgensen, T. Dyba, and B. A. Kitchenham, "Teaching Evidence-Based Software Engineering to University Students", Proceedings of the 11th IEEE International Software Metrics Symposium (METRICS '05), IEEE Computer Society, Sep. 2005, pp. 24-31, doi: 10.1109/METRICS.2005.46.

 ISBN: 978-1-61208-367-4

An Automated Signature Generation Method for Zero-day Polymorphic Worms Based on C4.5 Algorithm

Mohssen M. Z. E. Mohammed[1], Eisa Aleisa[2], Neco Ventura[3]
[1,2]College of Computer and Information Sciences, Al-Imam Muhammad Ibn Saud Islamic University, Riyadh, Saudi Arabia
[2]Department of Electrical Engineering, University of Cape Town, Rondebosch, South Africa
m_zin44@hotmail.com, aleisa@ccis.imamu.edu.sa, neco@crg.ee.uct.ac.za

***Abstract*— Polymorphic worms are considered as the most critical threats to the Internet security, and the difficulty lies in changing their payloads in every infection attempt to avoid the security systems. In this paper, we propose an accurate signature generation system for zero-day polymorphic worms. We have designed a novel double-honeynet system, which is able to detect zero-day polymorphic worms that have not been seen before. To generate signatures for polymorphic worms, we have two steps. The first step is the polymorphic worms sample collection, which is done by the double-honeynet system. The second step is the signature generation for the collected samples, which is done by a decision tree algorithm (C4.5 algorithm). The main goal for this system is to get accurate signatures for Zero-day polymorphic worm.**

Keywords- Honeynet; Polymorphic; Worms; Machine Learning; Algorithm.

I. INTRODUCTION

Due to the enormous threat from the worms, many efforts have been taken previously to tackle worms by detecting and preventing them. Later in this paper, the relevant works are discussed. However, in this section, internet worm defense methods and their limitations are mentioned in brief.

One avenue to deal with worms is prevention. We usually know that prevention is better than cure. Since worms need to exploit software defects, by eliminating all software defects we could eradicate worms. While theoretically this seems to be easy, the reality finds this as an almost impossible goal. Although significant progress has been made on software development, testing, and verification, empirical evidence [1][12] suggests that we are still far from producing defect-free software.

Another avenue to solve the worm problem is containment. Containment systems accept that software has defects that can be exploited by worms, and they strive to contain a worm epidemic to a small fraction of the vulnerable machines. The main challenge in designing containment systems is that they need to be completely automatic, because worms can spread far faster than humans can respond [1]. Recent works on automatic containment [14][15] have explored network-level approaches. These rely on heuristics to analyze network traffic and derive a packet classifier that blocks or rate-limits forwarding of worm packets.

It is hard to provide guarantees on the rate of false positives and false negatives with these approaches because there is no information about the software vulnerabilities exploited by worms at the network level. False negatives allow worms to escape containment, while false positives may cause network outages by blocking normal traffic. We believe that an automatic containment systems will not be widely deployed unless they have a negligible false positive rate.

It should be noted here that dealing with the prevention mechanisms is out of the scope of this paper because our work mainly focuses on containment mechanism of the worms.

We use a supervised Machine Learning (ML) algorithm [16] to generate signatures for polymorphic worms. Supervised machine learning is the search for algorithms that reason from externally supplied instances to produce general hypotheses, which then make predictions about future instances. In other words, the goal of supervised learning is to build a concise model of the distribution of class labels in terms of predictor features. There are several applications for ML, the most significant of which is data mining. People are often prone to making mistakes during analyses or, possibly, when trying to establish relationships between multiple features. This makes it difficult for them to find solutions to certain problems. Machine learning can often be successfully applied to these problems, improving the efficiency of systems and the designs of machines. Every instance in any dataset used by machine learning algorithms is represented using the same set of features. The features may be continuous, categorical or binary. If instances are given with known labels (the corresponding correct outputs) then the learning is called supervised [16], in contrast to unsupervised learning [16], where instances are unlabeled. By applying these unsupervised (clustering) algorithms, researchers hope to discover unknown, but useful, classes of items. Another kind of machine learning is reinforcement learning [16]. The training information provided to the learning system by the environment (external trainer) is in the form of a scalar reinforcement signal that constitutes a measure of how well the system operates. The learner is not told which actions to take, but rather must discover which actions yield the best reward, by trying each action in turn [16].

This paper is organized as follows: After Section I, Section II gives an introduction to decision tree algorithms.

 ISBN: 978-1-61208-367-4

Section III discusses the related works regarding automated signature generation systems. Section IV talks about the preliminaries of worms and their attacks. Section V discusses our Double-Honeynet system. Section VI introduces the proposed C4.5 algorithm. Section VII concludes the paper.

II. OVERVIEW FOR DECISION TREES

Decision trees classify instances by sorting them down the tree from the root to some leaf node, where:

- Each internal node specifies a test of some attribute.
- Each branch corresponds to a value for the tested attribute.
- Each leaf node provides a classification for the instance.

Figure 1 is an example of a decision tree for the training set of Table I.

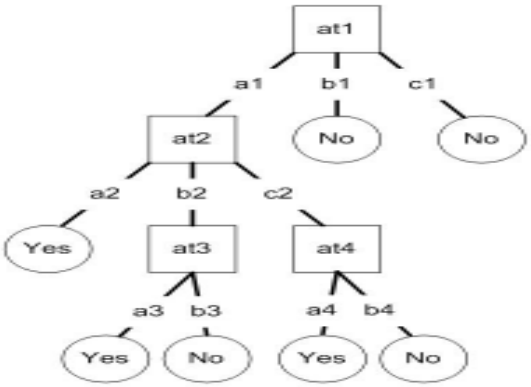


Figure 1. A decision tree.

TABLE I. TRAINING SET

at1	at2	at3	at4	Class
a1	a2	a3	a4	Yes
a1	a2	a3	b4	Yes
a1	b2	a3	a4	Yes
a1	b2	b3	b4	No
a1	c2	a3	a4	Yes
a1	c2	a3	b4	No
b1	b2	b3	b4	No
c1	b2	b3	b4	No

Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance [3].

Here, we give some explanation of Figure 1. The instance ⟨at1 = a1, at2 = b2, at3 = a3, at4 = b4⟩ would sort to the nodes: at1, at2, and finally at3, which would classify the instance as being positive (represented by the values "Yes"). The problem of constructing optimal binary decision trees is an NP-complete problem and thus theoreticians have searched for efficient heuristics for constructing near-optimal decision trees.

The feature that best divides the training data would be the root node of the tree. There are numerous methods for finding the feature that best divides the training data, such as information gain and gini index. While myopic measures estimate each attribute independently, ReliefF algorithm estimates them in the context of other attributes. However, a majority of studies have concluded that there is no single best method. Comparison of individual methods may still be important when deciding which metric should be used in a particular dataset. The same procedure is then repeated on each partition of the divided data, creating sub-trees until the training data is divided into subsets of the same class.

Below, we present a general pseudo-code for building decision trees.

```
Check for base cases
        For each attribute a
         Find the feature that best
         divides the training data such as
         information gain from splitting on a
 Let a_best be the attribute with the
 highest normalized information gain
         Create a decision node node that
         Splits on a_best
 Recurse on the sub-lists obtained by
 splitting on a_best and add those
 nodes as children of node
```

A decision tree, or any learned hypothesis h is said to overfit training data if another hypothesis h′ exists that has a larger error than h when tested on the training data, but a smaller error than h, when tested on the entire dataset. There are two common approaches that decision tree induction algorithms can use to avoid over-fitting training data which are [3]:

- Stop the training algorithm before it reaches a point at which it perfectly fits the training data,
- Prune the induced decision tree. If the two trees employ the same kind of tests and have the same prediction accuracy, the one with fewer leaves is usually preferred.

The most straightforward way of tackling over-fitting is to pre-prune the decision tree by not allowing it to grow to its full size. Establishing a non-trivial termination criterion such as a threshold test for the feature quality metric can do

 ISBN: 978-1-61208-367-4

that. Decision tree classifiers usually employ post-pruning techniques that evaluate the performance of decision trees, as they are pruned by using a validation set. Any node can be removed and assigned the most common class of the training instances that are sorted to it. Elomaa [16] concluded that there is no single best pruning method.

Even though the divide-and-conquer algorithm is quick, efficiency can become important in tasks with hundreds of thousands of instances. The most time consuming aspect is sorting the instances on a numeric feature to find the best threshold t. This can be expedited if possible thresholds for a numeric feature are determined just once, effectively converting the feature to discrete intervals, or if the threshold is determined from a subset of the instances. Elomaa and Rousu [16] stated that the use of binary discretization with C4.5 [3] needs about the half training time of using C4.5 multi splitting. In addition, according to their experiments, multi-splitting of numerical features does not carry any advantage in prediction accuracy over binary splitting.

Decision trees use splits based on a single feature at each internal node, so that they are usually univariate. In fact, most decision tree algorithms cannot perform well with problems that require diagonal partitioning. The division of the instance space is orthogonal to the axis of one variable and parallel to all other axes. Therefore, the resulting regions after partitioning are all hyperrectangles. However, there are a few methods that construct multivariate trees. S. B. Kotsiantis [16] presented Zheng's work, who improved the classification accuracy of the decision trees by constructing new binary features with logical operators such as conjunction, negation, and disjunction. In addition, Zheng created at-least M-of-N features. For a given instance, the value of an at o trees. In this model, new features are computed as linear combinations of the previous ones.

In the fact, decision trees can be significantly more complex representation for some concepts due to the replication problem. A solution to this problem is using an algorithm to implement complex features at nodes in order to avoid replication. In [16], S. B. Kotsiantis discussed Markovitch and Rosenstein work, and they presented the FICUS construction algorithm, which receives the standard input of supervised learning as well as a feature representation specification, and uses them to produce a set of generated features. While FICUS is similar in some aspects to other feature construction algorithms, its main strength is its generality and flexibility. FICUS was designed to perform feature generation given any feature representation specification complying with its general purpose grammar.

III. RELATED WORKS

Honeypots are an excellent source of data for intrusion and attack analysis. Levin et al. [4] described how Honeynet can be used to assist the system administrator in identifying malicious traffic on an enterprise network and how Honeypot-extracts with details of worm can be analyzed to generate detection signatures. The signatures are generated manually.

One of the first systems proposed was Honeycomb developed by Kreibich and Crowcroft [5]. Honeycomb generates signatures from traffic observed at a Honeypot via its implementation as a Honeyd plugin. The Longest Common Substring (LCS) algorithm, which looks for the longest shared byte sequences across pairs of connections, is at the heart of Honeycomb. Honeycomb generates signatures consisting of a single, contiguous substring of a worm's payload to match all worm instances. These signatures, however, fail to match all polymorphic worm instances with low false positives and low false negatives.

Kim and Karp [6] described the Autograph system for automated generation of signatures to detect worms. Unlike Honeycomb, Autograph's inputs are packet traces from a DMZ (demilitarized zone) that includes benign traffic. Content blocks that match "enough" suspicious flows are used as input to COPP [6], an algorithm based on Rabin fingerprints that searches for repeated byte sequences by partitioning the payload into content blocks. Similar to Honeycomb, Auto-graph generates signatures consisting of a single, contiguous substring of a worm's payload to match all worm instances. These signatures, unfortunately, fail to match all polymorphic worm instances with low false positives and low false negatives.

Singh et al. [7] described the Earlybird system for generating signatures to detect worms. This system measures packet-content prevalence at a single monitoring point, such as a network DMZ. By counting the number of distinct sources and destinations associated with strings that repeat often in the payload, Earlybird distinguishes benign repetitions from epidemic content. Earlybird, also like Honeycomb and Autograph, generates signatures consisting of a single, contiguous substring of a worm's payload to match all worm instances. These signatures, however, fail to match all polymorphic worm instances with low false positives and low false negatives.

New content-based systems, like Polygraph [8], Hamsa [10] and LISABETH [11], have been deployed. All these systems, similar to our system, generate automated signatures for polymorphic worms based on the following fact: there are multiple invariant substrings that must often be present in all variants of polymorphic worm payloads even if the payload changes in every infection. All these systems capture the packet payloads from a router, so in the worst case, these systems may find multiple polymorphic worms but each of them exploits a different vulnerability from each other. So, in this case, it may be difficult for the above systems to find invariant contents shared between these polymorphic worms because they exploit different vulnerabilities. The attacker sends one instance of a polymorphic worm to a network, and this worm in every infection automatically attempts to change its payload to generate other instances. So, if we need to capture all polymorphic worm instances, we need to give a polymorphic worm, chance to interact with hosts without affecting their performance. So, we propose a new detection method "*Double-honeynet*" to interact with polymorphic

 ISBN: 978-1-61208-367-4

worms and collect all their instances. The proposed method makes it possible to capture all worm instances and then forward these instances to the Signature Generator which generates signatures, using a particular algorithm.

An Automated Signature-Based Approach against Polymorphic Internet Worms by Tang and Chen [9] described a system to detect new worms and generate signatures automatically. This system implemented a Double-honeypot (inbound Honeypot and outbound Honeypot) to capture worms payloads. The inbound Honeypot is implemented as a high-interaction Honeypot, whereas the outbound Honeypot is implemented as a low-interaction Honeypot. This system has limitations. The outbound Honeypot is not able to make outbound connections because it is implemented as low-interaction honeypot which is not able to capture all polymorphic worm instances. Our system overcomes this disadvantage by using Double-honeynet (high-interaction Honeypot), which enables us to make unlimited outbound connections between them, so that we can capture all polymorphic worm instances.

All of the above works have used different algorithms to generate signatures for polymorphic worms, but there is no one in the above works using data mining algorithms to detect polymorphic worms. Data mining is a new technology and has successfully applied on a lot of fields; the overall goal of the data mining process is to extract information from a data set and transform it into an understandable structure for further use. Data mining is mainly used for model classification and prediction. classification is a form of data analysis that extracts models describing important data classes. C4.5 [16] is one of the most classic classification algorithms on data mining. So, in this paper, we used C4.5 algorithm for polymorphic worm classification. The C4.5 algorithm can classified each type of polymorphic worm into group.

The objective of using C4.5 is to generate signatures for polymorphic worms.

The advantages of using C4.5 algorithm over the others algorithms is the C4.5 can generate an accurate signatures for polymorphic worms.

IV. PRELIMINARIES OF WORM AND WORM ATTACKS

In this section, we talk about worms, so that the readers can learn how worm can attack victim computers connected to the Internet.

Worms are basically computer programs that self-replicate without requiring any human intervention; especially, by sending copies of their code in network packets and ensuring the code is executed by the computers that receive it. When computers become infected, they spread further copies of the worm and possibly perform other malicious activities.

A. Worm Infection

Remotely infecting a computer requires coercing the computer into running the worm code. To achieve this, worms exploit low-level software defects, also known as vulnerabilities. Vulnerabilities are common in current software, because today's software is usually large, complex, and mostly written in unsafe programming languages. Several different classes of vulnerabilities have been discovered over the years. Currently, buffer overflows, arithmetic overflows, memory management errors, and incorrect handling of format strings, are among the most common types of vulnerabilities exploitable by worms.

While we should expect new types of vulnerabilities to be discovered in the future, the mechanisms used by worms to gain control of a program's execution should change less frequently. Currently, worms gain control of the execution of a remote program using one of three mechanisms: injecting new code into the program, injecting new control-flow edges into the program (e.g., forcing the program to call functions that should not be called), and corrupting data used by the program.

B. Spread of Internet Worms

After infecting a computer, worms typically use it to infect other computers, giving rise to a propagation process which has many similarities with the spread of human diseases.

The spread of the worm in its most basic sense depends mostly on how it chooses its victims. This not only affects the spread and pace of the worm network but also its survivability and persistence as cleanup efforts begin. Classically, worms have used random walks of the Internet to find hosts and attack. However, new attack models have emerged that demonstrate increased aggressiveness.

C. Components of Worm

There are five basic components of worm:

Reconnaissance. The worm network has to hunt out other network nodes to infect. This component of the worm is responsible for discovering hosts on the network that are capable of being compromised by the worm's known methods.

Attack Components. These are used to launch an attack against an identified target system. Attacks can include the traditional buffer or heap overflow, string formatting attacks, Unicode misinterpretations (in the case of IIS (Internet Information Server) attacks), and misconfigurations.

Communication Components. Nodes in the worm network can talk with each other. The communication components give the worms the interface to send messages between nodes or some other central location.

Command Components. Once compromised, the nodes in the worm network can be issued operation commands using this component. The command element provides the interface to the worm node to issue and act on commands.

Intelligence Components. To communicate effectively, the worm network needs to know the location of the nodes as well as characteristics about them. The intelligence portion of the worm network provides the information needed to be able to contact with other worm nodes, which can be accomplished in a variety of ways [21].

 ISBN: 978-1-61208-367-4

V. OUR HONEYNET SYSTEM

We propose a Double-honeynet system to detect new worms automatically. A key contribution of this system is the ability to distinguish worm activities from normal activities without the involvement of experts.

Figure 2 shows the main components of the Double-honeynet system. Firstly, the incoming traffic goes through the Gate Translator which samples the unwanted inbound connections and redirects the sample connections to Honeynet 1. The gate translator is configured with publicly-accessible addresses, which represent wanted services. Connections made to other addresses are considered unwanted and redirected to Honeynet 1 by the Gate Translator.

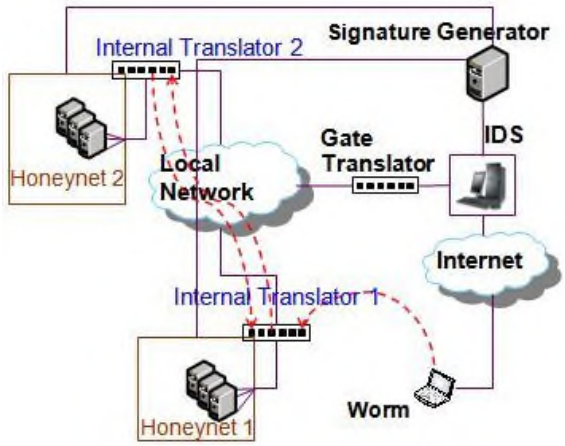


Figure 2. Double-honeynet system.

Secondly, once Honeynet 1 is compromised, the worm will attempt to make outbound connections. Each Honeynet is associated with an Internal Translator implemented in router that separates the Honeynet from the rest of the network. The Internal Translator 1 intercepts all outbound connections from Honeynet 1 and redirects them to Honeynet 2, which does the same, forming a loop.

Only packets that make outbound connections are considered malicious, and hence the Double-honeynet forwards only packets that make outbound connections. This policy is due to the fact that benign users do not try to make outbound connections if they are faced with non-existing addresses.

Lastly, when enough instances of worm payloads are collected by Honeynet 1 and Honeynet 2, they are forwarded to the Signature Generator component which generates signatures automatically using specific algorithms that will be discussed in the next section. Afterwards, the Signature Generator component updates the IDS database automatically by using a module that converts the signatures into Bro or pseudo-Snort format.

The above mentioned system was implemented by using Vmware Server 2 [13]. The details of the core implementation matters are out of the scope of this paper and were reported earlier; the readers are encouraged to read on the Double-honeynet architecture in our previously published work [13][17][18].

VI. C4.5 ALGORITHM

Motivation for Using C4.5 for Polymorphic Worms Detection

As it is known that a polymorphic worm can change its payload in every infection attempt, it is so difficult to know all instances of a polymorphic worm. In this paper, we use a well-known algorithm in classification problems, which is the C4.5. The advantage of using the C4.5 is polymorphic worm classifications.

We propose C4.5 algorithm to detects Zero-day polymorphic worms. The most well-know algorithm for building decision trees is the C4.5 [3]. C4.5 is an algorithm used to generate a decision tree developed by Ross Quinlan. This is an extension of Quinlan's earlier ID3 algorithm. The decision trees generated by C4.5 can be used for classification, and for this reason, C4.5 is often referred to as a statistical classifier.

C4.5 builds decision trees from a set of training data in the same way that ID3 does, using the concept of information entropy. The training data are a set $S = s_1, s_2, ...$ of already classified samples. Each sample, $s_i = x_1, x_2, ...$ is a vector where $x_1, x_2, ...$ represent attributes or features of the sample. The training data is augmented with a vector $C = c_1, c_2, ...$ where $c_1, c_2, ...$ represent the class to which each sample belongs.

At each node of the tree, C4.5 chooses one attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other. Its criterion is the normalized information gain (difference in entropy) that results from choosing an attribute for splitting the data. The attribute with the highest normalized information gain is chosen to make the decision. The C4.5 algorithm then recurses on the smaller subsists [3].

This algorithm has a few base cases:

- All the samples in the list belong to the same class. When this happens, it simply creates a leaf node for the decision tree saying to choose that class.
- None of the features provides any information gain. In this case, C4.5 creates a decision node higher up the tree using the expected value of the class.
- Instance of previously-unseen class encountered. Again, C4.5 creates a decision node higher up the tree using the expected value.

We should mention that machine learning algorithm are very slow in working, so in the future work we would like to use some of mathematical methods to enhances the our machine learning algorithm efficiency.

VII. CONCLUSION

In this paper, we have proposed an automated signature generation mechanism for zero-day polymorphic worms using a decision tree algorithm (C4.5 algorithm). In fact,

 ISBN: 978-1-61208-367-4

there are many other algorithms that have been proposed to generate signatures for zero-day polymorphic worms, but most of them have limitation to detect unknown pattern and also they have high computational complexity. Therefore, we have used C4.5 algorithm which overcomes these problems (detecting unknown pattern and computational complexity). One of the main advantages of machine learning algorithms is their great capacity to extract unknown and general information from a given data set (polymorphic worms samples) and its application on new data. The main goal of this paper was to use a machine learning technique (C4.5 algorithm) which can get better results than other algorithms such as string matching algorithms or similar others.

REFERENCES

[1] L. Spitzner, "Honeypots: Tracking Hackers," Addison Wesley Pearson Education: Boston, 2002.

[2] H. Bidgoli, "Handbook of Information Security," John Wiley & Sons, Inc., Hoboken, New Jersey.

[3] D. Gusfield, "Algorithms on Strings, Trees and Sequences,", Cambridge University Press: Cambridge, 1997.

[4] J. Levine, R. La Bella, H. Owen, D. Contis,and B. Culver, "The use of honeynets to detect exploited systems across large enterprise networks," Proc. of 2003 IEEE Workshops on Information Assurance, New York, Jun. 2003, pp. 92- 99.

[5] C. Kreibich and J. Crowcroft, "Honeycomb–creating intrusion detection signatures using honeypots," Workshop on Hot Topics in Networks (Hotnets-II), Cambridge, Massachusetts, Nov. 2003.

[6] H.-A. Kim and B. Karp, "Autograph: Toward automated, distributed worm signature detection," Proc. of 13 USENIX Security Symposium, San Diego, CA, Aug., 2004.

[7] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated worm fingerprinting," Proc. Of the 6th conference on Symposium on Operating Systems Design and Implementation (OSDI), Dec. 2004.

[8] J. Newsome, B. Karp, and D. Song, "Polygraph: Automatically generating signatures for polymorphic worms," Proc. of the 2005 IEEE Symposium on Security and Privacy, pp. 226 – 241, May 2005.

[9] Y. Tang, S. Chen, "An Automated Signature-Based Approach against Polymorphic Internet Worms," IEEE Transaction on Parallel and Distributed Systems, pp. 879-892, July 2007.

[10] Z. Li, M. Sanghi, Y. Chen, M.-Y. Kao and B. Chavez, "Hamsa: Fast Signature Generation for Zero-day Polymorphic Worms with Provable Attack Resilience," Proc. of the IEEE Symposium on Security and Privacy, Oakland, CA, May 2006.

[11] L. Cavallaro, A. Lanzi, L. Mayer, and M. Monga, "LISABETH: Automated Content-Based Signature Generator for Zero-day Polymorphic Worms," Proc. of the fourth international workshop on Software engineering for secure systems, Leipzig, Germany, May 2008.

[12] J. Nazario, "Defense and Detection Strategies against Internet Worms " Artech House Publishers (October 2003).

[13] M.M.Z.E. Mohammed, H.A. Chan, N. Ventura. "Honeycyber: Automated signature generation for zero-day polymorphic worms"; Proc. of the IEEE Military Communications Conference, MILCOM, 2008.

[14] Snort – The de facto Standard for Intrusion Detection/Prevention. Available: http://www.snort.org, 1 April 2012.

[15] Bro Intrusion Detection System. Available: http://www.bro-ids.org/, last accessed: 4 November 2013.

[16] S.B. Kotsiantis, "Supervised Machine Learning: A Review of Classification Techniques," Informatica 31 (2007) 249-268.

[17] M.M.Z.E. Mohammed and A.-S. K. Pathan. Automatic Defense against Zero-day Polymorphic Worms in Communication Networks, ISBN 9781466557277, CRC Press, Taylor & Francis Group, USA, 2013.

[18] M.M.Z.E. Mohammed and A.-S. K. Pathan, "Using Routers and Honeypots in Combination for Collecting Internet Worm Attacks," The State of the Art in Intrusion Prevention and Detection (Edited by Al-Sakib Khan Pathan), ISBN: 9781482203516, CRC Press, Taylor & Francis Group, USA, 2014 (To Appear).

 ISBN: 978-1-61208-367-4

On the Automation of Vulnerabilities Fixing for Web Application

Kabir Umar, Abu Bakar Sultan, Hazura Zulzalil, Novia Admodisastro, and Mohd Taufik Abdullah
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia, UPM
Serdang, Selangor, Malaysia
emails: kbrumar@yahoo.com, {abakar, hazura, novia, taufik}@upm.edu.my

Abstract -- **Testing Web applications for detection and fixing of vulnerabilities has become an indispensable task in web applications' development process. This task often consumes a lot of time, efforts and other resources. The research community have devoted considerable amount of efforts to address this problem by proposing many techniques for automated vulnerabilities detection and fix generation for web application. Many of these techniques can reliably detect vulnerabilities and generate fix(es), which can be applied to the web application's code, by the developer, for possible fixing of the vulnerabilities. Hence, the actual code modifications that fix the vulnerabilities is not automated and has to be carried out manually. To the best of our knowledge, none of the existing automated techniques is able to do this, and hence the actual fixing of the vulnerabilities is left for the human developer to handle. In this paper, we propose a novel framework for automatic vulnerabilities fixing for web application. We mimic evolutionary idea and employ Evolutionary Programming to evolve web applications whose fitness is evaluated based on their ability to survive test attacks. The reliability of the resulting vulnerabilities-free web application can be further enhanced by co-evolving test sets with generations of web applications in which the fitness of test attack is evaluated based on its ability to break web applications.**

Keywords-Web application; Automated Vulnerabilities Fixing; Evolutionary Programming; SQL Injection.

I. INTRODUCTION

In recent years, web applications and services have gained utmost popularity and acceptance in various fields of human endeavor. Unfortunately, these applications are often deployed with varied degrees of vulnerabilities that are exploitable by hackers through many types of attacks, which can result in unauthorized and, often, harmful transactions with the application, as well as its' underlying database [1][2][3]. The severe consequence of web application attacks is, perhaps, the reason why detection and fixing of these vulnerabilities has been among top priorities of both research communities, governments and industries [4][5].

For more than a decade now, many techniques were proposed in the literature, by different researchers from around the globe, for automated detection and generation of fix for these vulnerabilities. Although many of the proposed techniques can reliably detect vulnerabilities in a subject web application and generate possible fix, unfortunately, the non-trivial task of actual modification of the source code of the web application for fixing the detected vulnerabilities has to be done manually by the human developer. To the best of our knowledge, none of the existing techniques proposed in the literature has achieved complete automation of vulnerabilities fixing, in which actual code modifications to fix vulnerabilities is done automatically. Additional draw back of the manual code modification by applying the auto-generated fix is that, sometimes the resulting application may behave in an unexpected manner [6], thus compelling the developer to undo the changes and revert to the original application. Although in many cases applying the auto-generated fix does fix the vulnerabilities, this can only be ascertained through conducting another testing of the modified application. In order words, there is no guarantee that applying the auto-generated fix to the application will surely fix the detected vulnerabilities, another testing has to be done. This creates an unnecessary additional cost overhead because software testing consumes time, efforts and other vital resources [7][8].

Producing very secure web application is an important goal of software engineering [7] because doing so will greatly reduce or completely prevent attacks on web application and therefore, prevent losses incurred by governments, organizations and individuals. In this paper, we propose a novel framework for complete automation of vulnerabilities fixing for web application. The framework will make the actual source code modifications necessary to fix vulnerabilities. We explore the widely applied ideas of evolutionary computing [9][10][11] and use Evolutionary Programming (EP) to evolve web applications whose fitness is evaluated based on their ability to defend themselves from test attacks and pass legitimate input tests. The actual source code modifications will be achieved through evolutionary operation of mutation.

Furthermore, the reliability of the resulting vulnerabilities-free web application (V-freeWA) will be enhanced by co-evolving test sets along with generations of web applications. The test sets comprises test attacks whose fitness is evaluated based on its ability to break web applications and legitimate input test whose fitness is evaluated based on its ability to fail web applications. This creates competitive co-evolution between the population of programs and the population of test sets similar to what happens in nature between preys and predators, such as Antelopes and Tigers [11]. The main goal is to go beyond automated vulnerabilities detection and fix generation, and

 ISBN: 978-1-61208-367-4

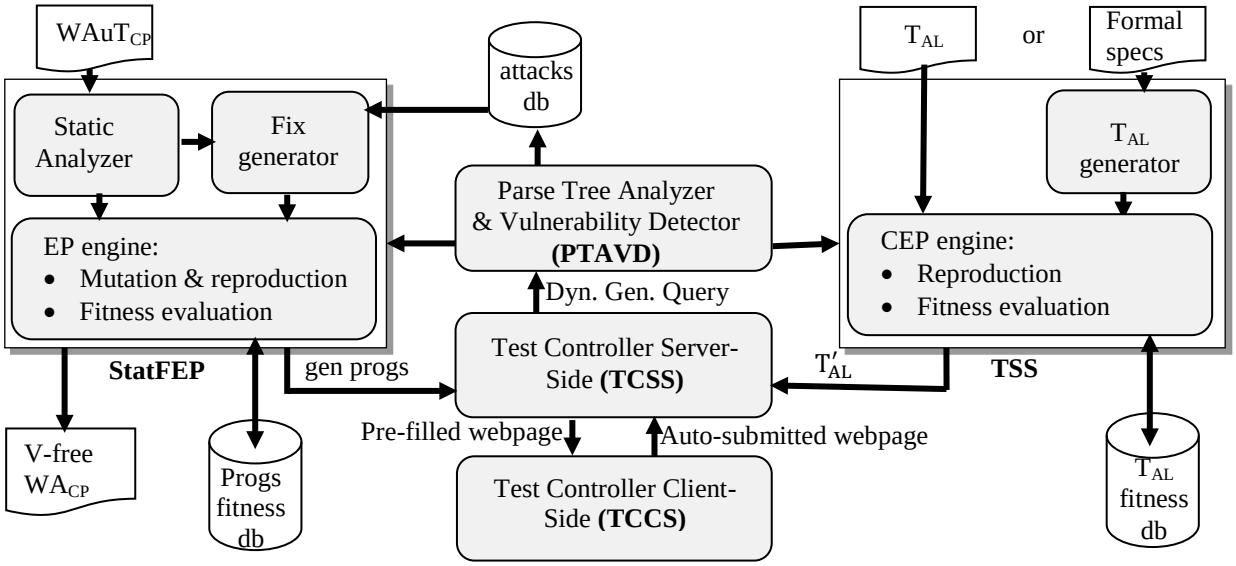


Figure 1. Framework for automated vulnerabilities fixing

have the actual task of code modification for vulnerabilities fixing fully automated. This challenging task is, perhaps, one of the long awaited automations in the field of software engineering. The remaining of this paper is organized as follows. Section II presents an overview of automated vulnerabilities detection and fix generation. Section III presents the proposed framework for automated vulnerabilities fixing. Section IV presents discussion, and Section V presents conclusion, future research work, as well as other possible research areas that can benefit from the framework.

II. OVERVIEW OF AUTOMATED VULNERABILITIES DETECTION AND FIX GENERATION

Many techniques were proposed in the literature for automated vulnerabilities detection and generation of possible fix(es) that can assist the developer to fix the vulnerable web application. Regrettably, the domain of web application vulnerabilities is very broad, diverse and exploitable by hackers through many types of attacks, thus making the task of automating vulnerabilities detection and fixing very challenging. The performance of the proposed techniques, in terms of vulnerabilities detection effectiveness, varies significantly from one category of vulnerabilities to another. While some of the techniques targeted specific category of vulnerabilities, others were proposed to handle considerably wide range of vulnerabilities, for example; Bau et al. [14] presented eight state-of-the-art black box vulnerabilities scanners that, on the average, each targets about six categories of vulnerabilities, namely, Cross Site Scripting (XSS), SQL Injection (SQLI), Cross Channel Scripting, Session Management, Cross Site Request Forgery (XSRF) and Information Leakage. Coincidentally, most of these vulnerabilities happen to be in the 2013 OWASP top 10 most dangerous web application security risk. Details about OWASP Top 10 project can be found in [4].

Interestingly, the techniques that addressed specific category of vulnerabilities also focus attention mostly within the OWASP Top 10 [4]; for example, [6][15][16] target SQL Injection (SQLI), [17][18] target Cross Site Scripting (XSS), [19] targets Buffer Overflow, [20] targets Configurations vulnerabilities, [21] targets Access Control vulnerabilities, [22] targets Session Management and Broken Authentication vulnerabilities, [23] targets Remote Code Execution, and [24] targets Logic vulnerabilities.

Although these techniques employs different software testing methods [7][8], such as static analysis, dynamic analysis, black box testing, penetration testing, mutation testing, search based testing, etc, and demonstrated diversity in their performance and effectiveness in vulnerabilities detection and fix generation, yet they almost have one thing in common, that is: “they were proposed to automate vulnerabilities detection and (in some cases) generate possible fix(es) in order to assist the developer to fix the vulnerable web application (under test)”. To the best of our knowledge, none of these techniques does the actual task of vulnerabilities fixing automatically.

III. AUTOMATED VULNERABILITIES FIXING

The proposed framework is for automatic vulnerabilities fixing for web applications. In this section, we present an overview of the components of the framework, highlight how they interacts, and highlight how fitness is evaluated for programs and test.

A. Components of the Framework

The framework comprises five main components, namely, Static analyzer, fix-generator, and EP engine (StatFEP), Test Set Selector (TSS), Test Controller Server

Side (TCSS), Test Controller Client Side (TCCS) and Parse Tree Analyzer and Vulnerability Detector (PTAVD). See Fig. 1 above.

1) StatFEP: Comprises three sub-components, namely, Static Analyzer, Fix-generator and EP engine. The static analyzer receives current page of web application under test ($WAuT_{CP}$) as input. It statically analyzes the webpage to determine all relevant database accessing points, sql generating statements, un-validated query-input variables, API method calls and parts of the webpage suitable for source code modification. The Fix-generator uses result of static analysis of $WAuT_{CP}$ and information from "attacks db" to generate smart fix(es) that targets likely vulnerabilities in $WAuT_{CP}$. The EP engine evolves population of $WAuT_{CP}$ as genetic programs (gen- progs) through single-parent reproduction and mutation operation. The mutation operation applies the auto-generated fix through source code modification guided by result of static analysis. In addition, the EP engine receives result of PTAVD to evaluate fitness of current individual program being tested and update "progs fitness db" accordingly. Lastly, this component monitors attainment of optimal solution and process stop condition.

2) TSS: The input to this component is test sets' search space T_{AL} or system's specifications of WAuT. T_{AL} is union of set of test attacks T_A and set of legitimate input tests T_L. The CEP-engine of TSS uses customized EP to evolve population of test sets T'_{AL} from the search space T_{AL}. Moreover, TSS uses result of PTAVD to evaluate fitness of individuals in T'_{AL} and update "T_{AL} fitness db" accordingly.

3) TCSS: This component receives current programs' individual (webpage) being tested from StatFEP, pre-fill the webpage with input data from current tests' individual in T'_{AL} (received from TSS), and forward the pre-filled webpage to TCCS. In addition, the component receives form submission (http POST request) from TCCS, intercepts and forward dynamically generated sql queries to PTAVD for analysis

4) TCCS: This component receives pre-filled webpage (current programs' individual being tested) from TCSS and auto-submit the page using http POST.

5) PTAVD: This component receives dynamically generated sql query from TCSS and performs syntax analysis for vulnerability detection by comparing syntax of current dynamic query with syntax of the same query generated using benign (verified legitimate) input from T_{AL}. Moreover, the component feeds information to StatFEP and TSS for appropriate fitness evaluations, and updates "attacks db" accordingly.

B. How it Works

To apply the proposed framework, a human tester might have confirmed the presence of at least one vulnerability in the Web Application under Test (WAuT) by subjecting it to test attacks (T_A) using appropriate testing method, such as applying tool for automated vulnerabilities detection. However, this is a very trivial and optional requirement. Nevertheless, since the framework is to auto-fix vulnerabilities, the presence of the vulnerabilities to fix could be confirmed first. Of course, if no vulnerabilities were detected in the WAuT, then there is no need to apply the framework.

Considering the nature of database access in web application, the StatFEP receives input of a page of the web application under test ($WAuT_{CP}$) at a time. The additional required input, received through TSS, is test sets T_{AL} comprising of test attacks and legitimate input tests. In addition, the TSS can receive system's specification of WAuT as input and generate T_{AL} accordingly. The $WAuT_{CP}$ is statically analyzed and represented as genetic program [9][12][13]. EP engine evolves the genetic programs. The CEP engine of TSS evolves the test sets T'_{AL}. The evolved programs in each generation are subjected to tests in T'_{AL}, by TCSS in collaboration with TCCS. During test execution, TCSS intercepts and forwards dynamic query to PTAVD for parse tree analysis, vulnerability detection, and functional correctness verification. Result of analysis is sent to EP engine for program's fitness evaluation, sent to CEP engine for test's fitness evaluation, and used to update "attacks db" accordingly. These fitness evaluations guide the evolution process to an optimal solution, i.e., V-freeWA_{CP}.

At each generation of programs, the operation of mutation, selection and reproduction is performed. Mutation applies the auto-generated fix to evolved programs. The auto-fix is generated by the fix generator module of StatFEP, with reference to information in "attacks db" and results of $WAuT_{CP}$ static analysis. Tournament is used to select programs with lowest fitness as parents. Single parent reproduction is employed to produce offspring [12]. Parents and offspring are combined to produce next generation. The optimal solution is found if an individual program has fitness of zero.

On the other hand, at each generation of test sets, only selection and reproduction is performed. Tournament is used to select test sets with highest fitness as parents. New test sets are randomly selected from test sets' search space to serve as offspring. Parents and offspring are combined to produce next generation of test sets.

C. Fitness Evaluation

1) Fitness of program: For easy reference in expressions, let $WAuT_{CP}$ be denoted by P_{or}, individual genetic program be denoted by P, and population of programs, consisting of n individuals P_1, P_2, …, P_n, be denoted by P_{pop}.

Fitness of P is evaluated based on semantic difference and syntax difference [9][28]. Semantic difference is a measure of how vulnerable, and how functionally incorrect, P is. Thus, we minimize semantic difference to ensure invulnerable and correct solution. On the other hand, syntax difference is a measure of how much P differs from P_{or} syntactically and structurally. Thus, we minimize syntax difference to ensure solution that respects the structure of P_{or}.

Definition 1: Given Q_d as the intercepted dynamic query generated by running P with a test set $t \in T'_{AL}$, Q_b as the same query generated from benign input, Syn(Q) as the syntax tree of query Q, and $T'_{AL}(P)$ as the set of assertions

(consisting of pairs (Q_d, Q_b)) after execution of P on all test sets $t \in \mathrm{T'_{AL}}$, the semantic difference of P is defined as follows:

$$f_{sem}(P) = \sum_{(Q_d,Q_b) \in \mathrm{T'_{AL}}(P)} astPT(Q_d, Q_b) \quad (1)$$

$$\text{Where,} \quad astPT(Q_d, Q_b) = \begin{cases} 0 & Syn(Q_d) = Syn(Q_b) \\ 1 & Syn(Q_d) \neq Syn(Q_b) \end{cases}$$

Definition II: Given $N(P_{or})$ as the number of nodes in syntax tree of $\mathrm{P_{or}}$, $N(P)$ as the number of nodes in syntax tree of P, and δ as the allowable safe nodes difference, the syntax difference of P is defined as follows:

$$f_{syn}(P) = \begin{cases} N(P) - N(P_{or}) - \delta & if\ N(P) > N(P_{or}) + \delta \\ N(P_{or}) - N(P) - \delta & if\ N(P) < N(P_{or}) - \delta \\ 0 & otherwise \end{cases} \quad (2)$$

Equations 1 and 2 are combined to define the fitness function of P. The goal is to minimize the fitness function.

$$f(P) = f_{sem}(P) + f_{syn}(P) \quad (3)$$

2) Test sets fitness function: The test sets T'_{AL} is consist of test attacks and legitimate input tests. The fitness of test attack is evaluated based on its ability to break P, while the fitness of legitimate input test is evaluated based on its ability to fail P. We define an expression that evaluates the fitness of any test set $t \in T'_{AL}$, where t can be a test attack or legitimate input test. The goal is to maximize fitness of t.

Definition II1: Given $Q_{P_i(t)}$ as the intercepted dynamic query generated by running $\mathrm{P_i}$ with test set $t \in \mathrm{T'_{AL}}$, Q_b as the same query generated from benign input, Syn(Q) as the syntax tree of query Q, and $P_{pop}(t)$ as the set of assertions (consisting of pairs $(Q_{P_i(t)}, Q_b)$) after execution of all individuals $P_i \in P_{pop}$ with test set $t \in \mathrm{T'_{AL}}$, the fitness of t is defined as follows:

$$f(t) = \sum_{P_i \in P_{pop}} astPT\left(Q_{P_i(t)}, Q_b\right) \quad (4)$$

$$\text{Where,} \ astPT\left(Q_{P_i(t)}, Q_b\right) = \begin{cases} 0 & Syn(Q_{P_i(t)}) = Syn(Q_b) \\ 1 & Syn(Q_{P_i(t)}) \neq Syn(Q_b) \end{cases}$$

IV. DISCUSSION

The fitness of program is evaluated based on its ability to defend attacks and pass legitimate inputs. The evolutionary operation of mutation is applied to make actual source code modifications. For simplicity, we seed the population of first generation of the genetic programs with many duplicate copies of $\mathrm{WAuT_{CP}}$. This is because we assume that $\mathrm{WAuT_{CP}}$ is both structurally, semantically and syntactically near V-free$\mathrm{WA_{CP}}$, considering the famous assumption that, software developers do not write programs at random [25]. The problem that we may encounter with this kind of seeding is lack of diversity in the first generation [12]. However, we can easily resolve this problem and achieve the required diversity by randomly applying the evolutionary operation of mutation to all members of the first generation.

Though our framework seems simple, unfortunately, the task of evolving vulnerabilities-free program is not an easy one. For instance, as a result of mutation operations the EP may evolve a too short program or too large program compared to the $\mathrm{WAuT_{CP}}$. This situation is not always trivial. Moreover, since we assumed that $\mathrm{WAuT_{CP}}$ is structurally and syntactically near the optimal solution, we don't want to have a solution that is too different (structurally and syntactically) from $\mathrm{WAuT_{CP}}$, because it might not be easily understood and maintained by the developer. This problem is handled by adding program size parameter to our fitness function (equation 2). We penalize too short or too large program, thereby minimizing structural and syntactic difference between $\mathrm{WAuT_{CP}}$ and the optimal solution.

Another problem that may be induced by EP mutation is degradation of functional properties of the $\mathrm{WAuT_{CP}}$ due to effects of source code modifications. This could, invariably, impair the correctness of resulting solution. Fortunately, the formulation of our semantic difference (equation 1) and composition of test sets $\mathrm{T_{AL}}$ can effectively take care of the situation. During program's fitness evaluation, the test attacks in $\mathrm{T_{AL}}$ tries to expose residual vulnerabilities, while the legitimate test inputs in $\mathrm{T_{AL}}$ tries to re-affirm functional correctness. This way, the correctness of resulting optimal solution is guaranteed.

An important factor that directly affects the reliability of the optimal solution produced by the framework is the quality of the test sets $\mathrm{T_{AL}}$ in terms of effectiveness and precision in revealing all residual vulnerabilities in, and re-affirming functional correctness of, $\mathrm{WAuT_{CP}}$. The emergence of V-free$\mathrm{WA_{CP}}$ that is able to defend all test attacks and pass all legitimate input tests in $\mathrm{T_{AL}}$ may not guarantee 100% vulnerabilities-free and functionally correct web application. This is true if the quality of the test sets $\mathrm{T_{AL}}$ is poor. One way we can tackle this problem and improve the quality of the test is to have large set of $\mathrm{T_{AL}}$ comprising of many diverse tests that target wide range of possible vulnerabilities and functional correctness. Obviously, this approach is very likely to reveal all vulnerabilities in $\mathrm{WAuT_{CP}}$, while maintaining its functional correctness. Unfortunately, using large set of $\mathrm{T_{AL}}$ will induce very high computational cost of fitness evaluation.

A more feasible approach is to employ a co-evolutionary mechanism in which population (of reasonable number) of test sets $\mathrm{T'_{AL}}$ (consisting of test attacks and legitimate input tests) is co-evolved along with every generation of the programs. To achieve this, the TSS component of the framework adopted the very large set $\mathrm{T_{AL}}$ as a test sets' search space from which generations of $\mathrm{T'_{AL}}$ are evolved using customized EP (CEP engine module). At each generation of test sets $\mathrm{T'_{AL}}$, fitness of individual test attack is evaluated based on its ability to break programs while

 ISBN: 978-1-61208-367-4

fitness of legitimate input test is evaluated based on its ability to fail programs. A test attack that breaks more programs is ranked more fit. A legitimate input test that fails more programs is ranked more fit. Fitness of both test attack and legitimate input test is evaluated using the same fitness function (equation 4). The idea is that, when test attack breaks a program, then the program is vulnerable, and when legitimate input test fails a program, then the program is functionally incorrect. Thus, we seek to maximize the fitness function of T'_{AL}. This approach will create competitive co-evolution between population of programs and population of test sets similar to what happens in nature between preys and predators, such as Antelopes and Tigers, in which an Antelope (prey) is rewarded for its ability to escape Tiger's hunt (predator), whereas a Tiger (predator) is rewarded for its ability to catch an Antelope (prey). In our co-evolutionary scenario, the programs are the preys while the test sets T'_{AL} are the predators. Thus, an evolved program is rewarded for defending against tests' hunting while a test is rewarded for being able to break or fail programs. As the co-evolutionary process go through generations, the population of test sets will go (hunting) after population of programs, while the population of programs try to survive by means of fitness, reproduction and mutation.

Although the co-evolutionary process can lead to emergence of highly reliable V-freeWA_{CP}, along the way, the process may suffer from problem of mediocre stable state and loss of gradient [26], which can occur when both population of preys and predators seem to positively evolve at each generation in an infinite circular pattern without any real improvement. This happens if the fitness evaluation of members of the co-evolving populations (programs and tests) is done without remembering what happened in previous generations. Fortunately, we can adopt Archives technique [26] to handle this problem. At each generation some individuals of programs and tests are stored into "prog fitness db" and "T_{AL} fitness db" respectively. The fitness of current generation is then based on interaction with the old individuals in the Archive, thus enabling the co-evolutionary process to remember history of past generations.

V. CONCLUSION AND FUTURE WORK

In addressing the problem of resolving web application's attacks and exploitations, this paper proposed a novel framework for automating vulnerabilities fixing for web application. The current techniques proposed in the literature are only capable of automating vulnerabilities detection and fix-generation while leaving the task of actual vulnerabilities fixing predominantly manual. We combine ideas of software testing, parse tree analysis, and evolutionary computing in a novel framework to achieve complete automation of the task. We have also shown how reliability of the resulting vulnerabilities-free web application can be further enhanced through co-evolution. The novel framework incorporates functional testing to guarantee correctness of the resulting optimal solution. As we progresses in this on-going research, we are optimistic in revealing and reporting very useful contributions toward automating vulnerabilities fixing, as well as advances in the field of software engineering.

Obviously, research on complete automation of vulnerabilities fixing for web application is at very preliminary stage for two reasons. First, to the best of our knowledge, this paper is the first to address the task. Second, the domain of web application's vulnerabilities is very large, broad and diverse [1][2][4][5]. Hence, there is room for a lots of future research activities. At the moment, we are planning the following:

A. Because the domain of web application vulnerabilities is very large, broad and diverse, we intend to scope our research to case study of SQL Injection vulnerabilities (SQLIVs). Our scope is so chosen for obvious reasons. First, SQLI has been the world's most serious web application security risk since 2004, as shown by OWASP Top 10 project reports of 2004, 2007, 2010, and 2013 [4] and CWE/SANS Top 25 Most Dangerous Software Errors [5]. Second, we are quite optimistic that applying the framework to successfully automate fixing SQLIVs, through well planned and documented experiment, should be sufficient to suggest the applicability of the framework for fixing other web application vulnerabilities related to source code.

B. We are planning to build our first prototype implementation of the framework in Java programming language to auto-fix vulnerabilities in web application designed in Java Server Pages (JSP) with MySQL as the backend database. The choice of JSP with MySQL backend is due to the fact that, most web applications are built in JSP, and MySQL database is, perhaps, the most widely used database server for the web.

Our novel framework can be applied (with slight modifications where necessary) to other research areas.

A. The framework can be used to automate fixing vulnerabilities in other non-web based database systems, such as Java database applications, which are equally liable to SQL injection attacks [27].

B. The framework can be adapted to automate Networks Vulnerabilities fixing through simulation. The idea is to have real network under test (NuT) simulated and evolved through generations. Fitness is then evaluated by subjecting the simulated networks to test attacks T_A.

Finally, we hope that our novel framework can benefit research community and lead to further research activities.

AKNOWLEDGEMENT

We acknowledge efforts and contributions of various authors whose work has benefited our research. We also acknowledge that this research received support from the Fundamental Research Grant Scheme (FRGS/2/2013/ICT01/UPM/02/8) awarded by Malaysian Ministry of Education to the Faculty of Computer Science and Information Technology at Universiti Putra Malaysia.

REFERENCES

[1] D. Watson and U. K. H. Project, “Web application attacks,” Journal of Network Security, vol. 2007, iss. 10, Oct. 2007, pp. 10–14, doi:10.1016/S1353-4858(07)70094-6.

[2] D. Gollmann, “Securing web applications,” Information Security Technical Report, ELSEVIER, vol. 13, no. 1, Jan. 2008, pp. 1–9, doi:10.1016/j.istr.2008.02.002.

[3] A. Garg and S. Singh, “A review on web application security vulnerabilities,” International Journal of Advanced Research in Computer Science and Software Engineering, (IJARCSSE), vol. 3, no. 1, 2003, pp. 222–226.

[4] OWASP, "OWASP top 10 project," https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project , [retrieved: May 2014].

[5] CWE, “CWE-SANS top 25 most dangerous software errors,” Common Weakness Enumeration, Http://cwe.mitre.org/ top25/. (http://cwe.mitre. Org/top25/), [retrieved: May 2014].

[6] J. M. Chen and C. L. Wu, "An automated vulnerability scanner for injection attack based on injection point," Proc. IEEE International Computer Symposium, IEEE Press, Tainan, Dec. 2010, pp. 113-118, doi:10.1109/COMPSYM.2010.5685537.

[7] L. Luo, "Software testing techniques," Class Report for 17-939A, Institute for Software Research International, Carnegie Mellon University, USA. (http://mcahelpline.com/tutorials/testing/testing.pdf).

[8] J. Irena, “Software testing methods and techniques,” The IPSI BgD Transactions on Internet Research, 2008, internetjournals.net.

[9] A. Arcuri, "Evolutionary repair of faulty software," Journal of Applied Soft Computing, ELSEVIER, vol. 11, iss. 4, June. 2011, pp. 3494–3514, doi:10.1016/j.asoc.2011.01.023.

[10] A. Arcuri, "On search based software evolution," Proc. IEEE 1st International Symposium on Search Based Software Engineering, IEEE Press, Windsor, May. 2009, pp. 39–42, doi:10.1109/SSBSE.2009.12.

[11] A. Arcuri, "On the automation of fixing software bugs," Proc. ACM 30th International Conference on Software Engineering, ACM, Leipzig, Germany, May. 2008, pp. 1003-1006, doi:10.1145/1370175.1370223.

[12] A. Abraham, "Evolutionary computation: from Genetic Algorithms to Genetic Programming," in Genetic Systems Programming: Theory and Experiences, Ecological Studies 185, N. Nedjah and A. Abraham, Springer, 2005, pp. 1-20.

[13] R. G. S. ASTHANA, "Evolutionary Algorithms and Neural Networks," in Soft Computing and Intelligent Systems: Theory and Applications, A volume in Academic Press Series in Engineering, N. K. Sinha, M. M. Gupta and L. A. Zadeh, ELSEVIER Inc, 2000, pp. 111-136.

[14] J. Bau, E. Bursztein, D. Gupta and J. Mitchell, "State of the art: automated black-box web application vulnerability testing," Proc. IEEE Symp. Security and Privacy (SP), IEEE Press, Oakland, CA, May. 2010, pp. 332-345, doi:10.1109/SP.2010.27.

[15] F. Dysart and M. Sherriff, "Automated fix generator for SQL injection attacks," Proc. IEEE 19th International Symposium on Software Reliability Engineering, (ISSRE), IEEE Press, Seattle, WA, Nov. 2008, pp. 311-312, doi:10.1109/ISSRE.2008.44.

[16] Z. Djuric, “A black-box testing tool for detecting SQL injection vulnerabilities,” Proc. IEEE Second International Conference on Informatics and Applications, (ICIA), IEEE Press, Lodz, Sept. 2013, pp. 216-221, doi:10.1109/ICoIA.2013.6650259.

[17] B. Qu, B. Liang, S. Jiang and C. Ye, "Design of automatic vulnerability detection system for web application program," Proc. IEEE 4th International Conference on Software Engineering and Service Science, (ICSESS), IEEE Press, Beijing, May. 2013, pp. 89-92, doi:10.1109/ICSESS.2013.6615262.

[18] G. Wassermann and Z. Su, "Static detection of Cross-Site Scripting vulnerabilities," Proc. ACM/IEEE 30th International Conference on Software Engineering, (ICSE), ACM/IEEE, Leipzig, May. 2008, pp. 171-180, doi:10.1145/1368088.1368112.

[19] A. Smirnov and T. Chiueh, "Automatic patch generation for Buffer Overflow attacks," Proc. IEEE Third International Symposium on Information Assurance and Security, (IAS), IEEE Press, Manchester, Aug. 2007, pp. 165-170, doi:10.1109/IAS.2007.87.

[20] B. Eshete, A. Villafiorita, K. Weldemariam and M. Zulkernine, "Confeagle: Automated analysis of Configuration vulnerabilities in web applications," Proc. IEEE 7th International Conference on Software Security and Reliability, (SERE), IEEE Press, Gaithersburg, MD, June. 2013, pp. 188-197, doi:10.1109/SERE.2013.30.

[21] F. Gauthier and E. Merlo, "Fast detection of Access Control vulnerabilities in PHP applications," Proc. IEEE 19th Working Conference on Reverse Engineering, (WCRE), IEEE Press, Kingston, ON, Oct. 2012, pp. 247-256, doi:10.1109/WCRE.2012.34.

[22] D. Huluka and O. Popov, "Root cause analysis of Session Management and Broken Authentication vulnerabilities," IEEE World Congress on Internet Security, IEEE Press, Guelph, ON, June. 2012, pp. 82-86.

[23] Y. Zheng and X. Zhang, "Path sensitive static analysis of web applications for remote code execution vulnerability detection," Proc. IEEE 35th International Conference on Software Engineering, (ICSE), IEEE Press, San Francisco, CA, May. 2013, pp. 652-661, doi:10.1109/ICSE.2013.6606611.

[24] V. Felmetsger, L. Cavedon, C. Kruegel and G. Vigna, "Toward automated detection of logic vulnerabilities in web applications," Proc. ACM 19th USENIX conference on Security, ACM, Berkeley, CA, USA, 2010, pp. 10-10.

[25] R. A. DeMillo, R. J. Lipton and F. Sayward, "Hints on test data selection: help for the practicing programmer," Computer, vol. 11, iss. 4, IEEE Press, 2006, pp. 34–41, doi:10.1109/C-M.1978.218136.

[26] C. D. Rosin and R. K. Belew, "New methods for competitive coevolution," in Evolutionary Computation, vol. 5, iss. 1, MIT Press Cambridge, MA, USA, 1997, pp. 1-29.

[27] C. Zhou and P. Frankl, "JDAMA Java database application mutation analyser," ACM Journal of Software Testing, Verification & Reliability, vol. 21, iss. 3, Sept. 2011, pp. 241-263, doi:10.1002/stvr.462.

[28] W. B. Langdon, M. Harman and Y. Jia, “Efficient multi-objective higher order mutation testing with genetic programming,” ELSEVIER Journal of Systems and Software, vol. 83, iss. 12, Dec. 2010, pp. 2416–2430, doi:10.1016/j.jss.2010.07.027.

 ISBN: 978-1-61208-367-4

An Approach for Cross-Site Scripting Detection and Removal Based on Genetic Algorithms

Isatou Hydara, Abu Bakar Md Sultan, Hazura Zulzalil, Novia Admodisastro
Dept. of Software Engineering and Information System
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
43400 UPM Serdang, Selangor, Malaysia
ishahydara@gmail.com, abakar@upm.edu.my, hazura@upm.edu.my, novia@upm.edu.my

***Abstract* – Software security vulnerabilities have led to many successful attacks on applications, especially web applications, on a daily basis. These attacks, including cross-site scripting, have caused damages for both web site owners and users. Cross-site scripting vulnerabilities are easy to exploit but difficult to eliminate. Many solutions have been proposed for their detection. However, the problem of cross-site scripting vulnerabilities present in web applications still persists. In this paper, we propose to explore an approach based on genetic algorithms that will be able to detect and remove cross-site scripting vulnerabilities from the source code before an application is deployed. The proposed approach is, so far, only implemented and validated on Java-based Web applications, although it can be implemented in other programming languages with slight modifications. Initial evaluations have indicated promising results.**

Keywords-cross-site scripting; genetic algorithm; software security; vulnerability detection; vulnerability removal.

I. INTRODUCTION

Security testing is becoming an important part of software development due to the numerous attacks that software applications encounter on a daily basis. Due to their dynamic nature, i.e., the changing of their content in real-time as a result of user input or of being reloaded, web applications are the most exposed to security attacks, such as cross-site scripting (XSS). Many research activities have been conducted to address problems related to XSS vulnerabilities since their discovery. Most of the approaches focused on preventing XSS attacks [1][2][3][4] or detecting XSS vulnerabilities [5][6][7][8] in web applications during software security testing. Few research activities have addressed their removal [9][10].

Software systems are usually deployed to the public with unexpected security holes. This is mainly due to the short time frame in which software are developed. Software project managers do not cater for security issues in their budgeting, scheduling and staffing their software development projects. Despite the fact that attention on software security is increasing, the progress on research for great solutions is slow**.** Notwithstanding that research on software security is very recent, effective solutions are in high demand due to the importance of creating software that is more secure and is less vulnerable to attacks.

In this paper, we propose a genetic algorithm-based approach for the detection and removal of XSS vulnerabilities in web applications. The rest of the paper is organized as follows: Section II gives a background of XSS and Genetic Algorithms. Section III reviews related research conducted on the problems of XSS. In Section IV, we describe our proposed approach and expected experimental results. Section V concludes the paper.

II. BACKGROUND

A. Cross-Site Scripting

Cross-site scripting vulnerabilities are a security problem that occurs in web applications. They are among the most common and most serious security problems affecting web applications [11][12]. They are a type of injection problems [12] that enable malicious scripts to be injected into trusted web sites. This is a result of a failure to validate input from the web site users. What happens is either the web site fails to neutralize the user input or it does it incorrectly [11], thus, opening an avenue for a host of attacks.

Successful XSS can result in serious security violations for both the web site and the user. An attacker can inject a malicious code into where a web application accepts user input, and if the input is not validated, the code can steal cookies, transfer private information, hijack a user's account, manipulate the web content, cause denial of service, and many other malicious activities [11][12].

Cross-site scripting attacks are of three types namely reflected, stored and DOM (Document Object Model)-based [11][12]. Reflected XSS is executed by the victim's browser and occurs when the victim provides input to the web site. Stored XSS attacks store the malicious script in databases, message forums, comments fields, etc. of the attacked server. The malicious script is executed by visiting users thereby passing their privileges to the attacker. Both reflected and stored XSS vulnerabilities can be found on either client side or server side codes. On the other hand, DOM-based XSS vulnerabilities are found on the client side. Attackers are able to collect sensitive or important information from the user's computer.

 ISBN: 978-1-61208-367-4

B. Genetic Algorithms

Genetic Algorithms (GAs) are a subset of Evolutionary Algorithms (EAs), which are metaheuristic optimization algorithms based on population and inspired by biology [13]. They employ natural evolution mechanisms, such as mutation, crossover, natural selection, and survival of the fittest [14] to find optimal solutions in a search space. GAs are different from other EAs in that they have a crossover (recombination) operation and use binary coding in bits or bit-strings to represent a population [14].

Genetic algorithms have many capabilities; they have been used in many areas of computer science, such as software testing [15] and intrusion detection in network security [16][17] and in many other fields as well. In our proposed research, we believe similar techniques used in intrusion detection can be employed in the detection of cross-site scripting vulnerabilities in web applications. Experimentation need to be carried out to investigate this possibility.

Similarly, GAs can be used to generate source code with proper encoding that will replace parts of a source that is found to contain XSS vulnerabilities. For this part, similar methods used in test case generation with genetic algorithms in software testing can be employed.

Genetic algorithms have proven to be good solutions to many software engineering problems, since their discovery. Their successful use in software security testing [8][18][19] and intrusion detection systems [16][17] gives us the hope that they will be useful in detecting and removing XSS security vulnerabilities in Java-based Web applications.

III. RELATED WORK

Avancini and Ceccato [19] investigated the integration of taint analysis with genetic algorithms as an approach in software security testing of web applications. Their method showed some improvement in capturing XSS vulnerabilities and using them as a test case in security testing. They also implemented the integration of static taint analysis, genetic algorithm and constraint solving to automatically generate test cases that detect cross-site scripting vulnerabilities [18]. Their implementation focused only on reflected XSS in PHP code. The results seem promising. However, the fitness function of the genetic algorithm needs to be strengthened and the model tested in a wider range of software systems.

Duchene et al. [8] proposed an approach that combined model inference and evolutionary fuzzing to detect XSS vulnerabilities. Their approach used model inference to obtain a state model of the system under test and then used genetic algorithm to generate test input sequences, which enabled the detection of vulnerabilities. An explanation of their technique indicated it would prove successful when implemented on real world applications.

Lwin and Hee [10] proposed a solution that is able to remove XSS vulnerability from web applications before they can be exploited by hackers. The approach works in two phases. First, it uses static analysis to identify potential XSS vulnerabilities in application source codes. Secondly, it uses pattern matching techniques to come up with appropriate escaping mechanisms to prevent input values from causing script execution.

Researchers have also proposed tools that address the problem of XSS. BIXSAN [20] and L-WMxD [21] are two examples of such tools developed to tackle the XSS problem. BIXSAN filters out harmful HTML content and removes the non-static tags in the HTML page. It has been tested on many web browsers and shown to successfully prevent XSS attacks. L-WMxD, on the other hand, works on Webmail services to detect the presence of XSS vulnerabilities. The tool has been tested on real-world Webmail applications with some limitations and the results seem promising.

IV. PROPOSED APPROACH

The solution being proposed uses a genetic algorithm-based approach in the detection and removal of XSS vulnerabilities in Web applications. The proposed solution is in three components. The first component involves converting the source codes of the applications to be tested to Control Flow Graphs (CFGs) using the White Box Testing techniques, where each node will represent a statement and each edge will represent the flow of data from node to node. A static analysis tool, PMD [22], is used in this task. The second component focuses on detecting the vulnerabilities in the source codes whiles the third component concentrates on their removal.

The main idea behind our approach is to formulate the security testing for XSS vulnerabilities as a search optimization problem. GAs have proved successful in the generation of minimal number test cases to uncover as many flows as possible in source codes [23]. In the same way, we can use GAs to detect as many XSS vulnerabilities as possible with a minimal number of test cases.

The main contributions of this work are:

- The detection of XSS vulnerabilities in the source code of web applications using a GA approach
- The removal of detected XSS vulnerabilities from the source code of web applications
- The automation of the XSS vulnerabilities detection and removal approach

A. Taint Analysis

Taint analysis is a White Box testing technique that tracks tainted or untainted status of variables throughout the control flow of an application and determines if a sensitive statement is used without validation [10][19][24]. For XSS vulnerabilities, a tainted variable refers to inputs from user or database, and print statements that append a string into a web page.

To perform a complete analysis of an application source code, we need to follow the White Box testing coverage criteria, such as statement coverage, branch coverage, or path coverage. We choose path coverage criterion because it encompasses the previous two. However, it is generally impossible to cover all paths of the source code in testing. Therefore, we select a subset of the paths that interest us; the vulnerable paths whose execution will reveal XSS vulnerabilities. These are the paths where an input is executed without validation.

 ISBN: 978-1-61208-367-4

B. The Genetic Algorithm

Basically, a genetic algorithm consists of the following steps:

- Step 1: Create an initial population of candidate solutions
- Step 2: Compute the fitness value of each candidate
- Step 3: Select all the candidates that have their fitness values above or on a threshold
- Step 4: Make changes to each of the selected candidates using genetic operators, e.g., crossover and mutation
- Step 5: Repeat from step 2 until solution is reached or exit criteria is met.

The above steps are converted into a pseudocode, as shown in Figure 1 below.

```
population = generate_random_population();
        for(T in vulnerable paths) {
  while(T not covered AND attempt < max_try) {
        selection = select(population);
      offspring = crossover(selection);
      population = mutate(offspring);
          attempt = atempt + 1;
              }
            }
```

Figure 1. Genetic Algorithm Pseudocode.

1) Representation

The most common form of representing or encoding chromosomes in GA is using binary format. However, using binary format in XSS vulnerabilities detection would be very complex since the chromosomes represent patterns of real strings that serve as inputs while testing. Therefore, we decided to use natural numbers as the encoding scheme in our GA.

2) Initial Population

The GA population refers to the set of possible solutions for the problem to be solved. These possible solutions are generally referred to as chromosomes. In this work, the initial population is a set of test data that is generated according to the path coverage criterion, as stated in Section IV A. Since Gas deal with large search space, we will use a large population size of at least 100. After the initial population is selected, each individual chromosome is evaluated for possible inclusion in the next generation based on the fitness function.

3) Fitness Function

The fitness function is a measure of how good a chromosome is at solving the problem under consideration. So, a chromosome has a higher fitness value if it is closer to solving the problem. For our work, the fitness function evaluates the vulnerable paths that a test case needs to follow in order to reveal the presence of XSS vulnerabilities. It calculates the percentage of branches covered by an input traversing a vulnerable path and assigns a value. For example, if an input traverses all the branches of a vulnerable path, it means it has covered 100% of the branches and is assigned the value 1. If it traverses 70%, it is assigned the value 0.7 and so on. Hence, our fitness function is

$$F(x) = ((Cpaths\% + Diff) * XSSp\%)/100.$$

F(x): the fitness for an individual chromosome
Cpath%: the percentage of branches covered
Diff: the difference between the traversed and the targeted paths
XSSp%: the percentage of the XSS patterns file that the GA uses to cover a test path

4) Selection

For each iteration of the GA, a sample of chromosomes is selected for evaluation for possible inclusion in the next generation. There are different selection techniques for GA and in this work we choose the roulette wheel selection technique. It is a popular technique whereby the probability of selecting a chromosome for the next generation is proportional to its fitness function value. Two chromosomes (parents) are selected randomly based their fitness function values and subjected to crossover and mutation methods in order to produce new chromosomes (offspring) for the next generation.

5) Crossover

In the crossover operation, as shown in Figure 2, two chromosomes are combined to form other chromosomes in the hope that the new ones will be better than the parent chromosomes. We use uniform crossover whereby the parent chromosomes contribute to the new offspring according to a specific crossover probability. We use a probability of 0.5 for the crossover operation. This is to give a fifty percent chance for half of the chromosomes to undergo changes while the other half proceeds to the next generation without undergoing any changes. This is because some chromosomes may already contain good genes and need not be changed.

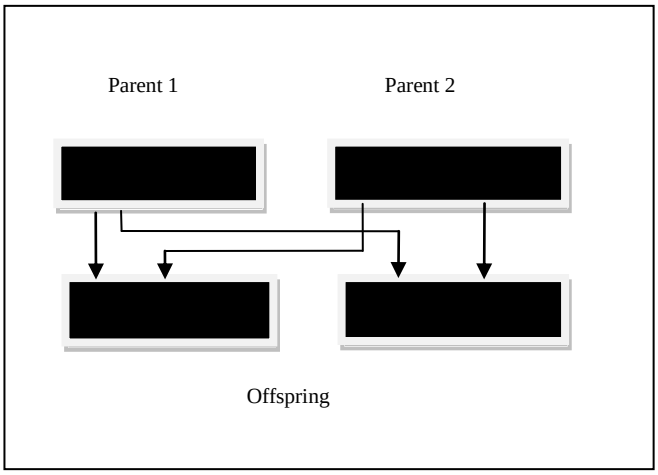


Figure 2. Crossover

6) Mutation

The mutation operator is performed on the offspring after the crossover. It alters the chromosome values according to a specific mutation probability. It helps to guarantee that the entire search space is search, given enough time. It also helps to restore lost information or add more information to the population. A low mutation probability of 0.2 is used.

C. The cross-site scripting removal Stage

Once the XSS vulnerabilities are detected in the source code, the removal stage begins. The OWASP's ESAPI (Enterprise Security API) security mechanisms [25] are followed to remove the detected XSS vulnerabilities. The lines of code where the XSS vulnerabilities are located will be identified. Then, we determine which of the ESAPI escaping rules can be applied to replace those lines of code without compromising their functionality. Finally, we generate the secure codes of the escaping statements and put them in place of the vulnerable statements, using these ESAPI escaping rules:

- **Rule#1:** Use HTML entity escaping for the untrusted data referenced in an HTML element, for example, <body> htmlEscape(untrusted_data) </body>, where ‘‘htmlEscape()’’ is the HTML entity escaping method
- **Rule#2:** Use HTML attribute escaping for the untrusted data referenced as a value of a typical HTML attribute such as name and value, for example, <input value=‘htmlAttrEscape(untrusted_data)’>, where ‘‘htmlAttrEscape()’’ is the HTML attribute escaping method
- **Rule#3:** Use JavaScript escaping for the untrusted data referenced as a quoted data value in a JavaScript block or an eventhandler, for example, <bodyonload=‘‘x=‘javascriptEscape(untrusted_data)’’’>, where ‘‘javascriptEscape()’’ is the JavaScript escaping method
- **Rule#4:** Use CSS escaping for the untrusted data referenced as a value of a property in a CSS style, for example,<table style= ‘‘width: cssEscape(untrusted_data)’’>, where ‘‘cssEscape()’’ is the CSS escaping method
- **Rule#5:** Use URL escaping for the untrusted data referenced as a HTTP GET parameter value in a URL, for example, <a href=‘http://www.site.com?name=urlEscape(untrusted_data)’>, where ‘‘urlEscape()’’ is the URL escaping method

Figures 3 and 4 [10] present the encoding mechanism of ESAPI.

```
public class Login extends HttpServlet {
       public void doGet (HttpServletRequest req,
     HttpServletResponse resp){

1  String memID = req.getParameter("id");
2  String pwd = req.getParameter("password");
3  String html = "<HTML><BODY><h1>"; //HTML structure
     . . .//perform database access & input validation
4  if(LoginValid) {
       //retrieve Member Name from database
5       String name = rs.getString("LastName");
6       html += "Welcome "+ name + "! </h1>";
   }
   else {
7       memID = "S123456";
8       html += "Welcome Guest! </h1>";
   }

9  memID = memID.substring(0,7);

10 html += "<input type='hidden' name='member_id'
                      value= '"+memID+"'>";
11 out.print(html);
       . . .
}
```

Figure 3. A potentially vulnerable code.

```
import org.owasp.esapi.ESAPI;
public class Login extends HttpServlet {
     . . .
   String memID = req.getParameter("id");
   String pwd = req.getParameter("password");
   String html = "<HTML><BODY><h1>";

   if(LoginValid) {
        String name = rs.getString("LastName");

        //html += "Welcome "+ name + "! </h1>";
        html += "Welcome "+
                  ESAPI.encoder().encodeForHTML(name) + "! </h1>";
   }
   else {
        memID = "S123";
        html += "Welcome Guest! </h1>";
   }

   memID = memID.substring(0,7);

   //html += "<input type='hidden' name='member_id'
                    value= '"+memID+"'>";
   html += "<input type='hidden' name='member_id' value='" +
              ESAPI.encoder().encodeForHTMLAttribute(memID)+ "'>";
   out.print(html);
```

Figure 4. Code secured with ESAPI security API.

D. Evaluation

The above approach is being implemented in a prototype and will be evaluated for its effectiveness and performance. The data needed for the experiments on this research will be full source codes of Web applications. Source codes of complete open-source Java-based large Web applications will be used as experimentation data. These projects will be collected mainly from the Source Forge site [26], as they are freely available.

All development is being implemented with the Eclipse IDE using the Java Programming Language. The JGAP (Java Genetic Algorithm Package) engine [27] is integrated into the Eclipse IDE as a library for the easy usage of its Genetic Algorithm operators. Java-based static analysis tool, PMD, is used to generate the CFG of the application files to be tested.

For this research, we used Java as the programming language on which to test our approach. Although most of the existing web sites are built with PHP, JavaScript and other similar scripting languages, there are many web sites built using Java Server Pages. These websites are also exposed to the cross-site scripting problem. Besides, most of the existing research works conducted on cross-site scripting were implemented using languages other than Java; hence, the focus on Java.

E. Expected Results

This research is expected to produce a new approach to detecting and removing XSS vulnerabilities in Java-based web applications. This approach will be an improvement based on the combination of two previously proposed approaches [10][19]. The first approach uses genetic algorithms to detect reflected XSS vulnerabilities only but does not remove them. The second approach is able to detect and remove both reflected and stored XSS vulnerabilities using pattern matching technique, but not DOM-based XSS. By combining them, this research will be able to use an enhanced genetic algorithm to detect and remove not only the same vulnerabilities but also DOM-based XSS vulnerabilities, which are not covered by both approaches. A Java-based tool has been developed to automate this approach. Furthermore, we expect this new approach to benefit web application developers by enabling them to easily test their source codes and get rid of many XSS vulnerabilities before deployment of their systems. This in turn will benefit any user who accesses such web systems by protecting them from malicious attacks.

F. Limitations

The limitations of this work are listed below:

1. Since this work makes use of static analysis, it also suffers its limitations. Therefore, the approach will fail to detect XSS vulnerabilities whose paths cannot be identified by static analysis in the source code.
2. The vulnerabilities removal module of the approach uses the OWASP ESAPI's escaping API only. Therefore, XSS vulnerabilities that are not defined in the context of this API are out of the scope of this work.
3. The approach is so far only implemented on Java Server Pages Web applications. However, the approach can be used with other programming languages.

V. Conclusion and Future Work

In this paper, we presented a genetic algorithm-based approach for XSS detection and removal. Cross-site scripting is a major security problem for web applications. It can lead to account or web site hijacking, loss of private information, and denial of service, all of which victimize the site users. Our proposed approach is an improvement based on two previously proposed approaches. It uses better and improved GA operators to help in the detection and removal of XSS vulnerabilities as well as including all the three types of XSS. Our next step on this progressive work is to fully evaluate and validate the proposed approach. A prototype tool has been developed to automate this process. Preliminary evaluation show promising results. We will continue to test the approach on real world Web applications and also improve the prototype tool. We expect our approach to be able to detect and remove the majority of XSS vulnerabilities, if not all, in real world Web applications.

Acknowledgment

This work was supported by the Malaysian Ministry of Education under the Fundamental Research Grant Scheme (FRGS 08-02-13-1368).

References

[1] P. Sharma, R. Johari, and S. S. Sarma, “Integrated approach to prevent SQL injection attack and reflected cross site scripting attack,” Int. J. Syst. Assur. Eng. Manag., vol. 3, no. 4, Sep. 2012, pp. 343–351.

[2] Y. Sun and D. He, “Model Checking for the Defense against Cross-Site Scripting Attacks,” in 2012 International Conference on Computer Science and Service System, 2012, pp. 2161–2164.

[3] M. Van Gundy and H. Chen, “Noncespaces: Using randomization to defeat cross-site scripting attacks,” Comput. Secur., vol. 31, no. 4, Jun. 2012, pp. 612–628.

[4] T. Scholte, W. Robertson, D. Balzarotti, and E. Kirda, “Preventing Input Validation Vulnerabilities in Web Applications through Automated Type Analysis,” in 2012 IEEE 36th Annual Computer Software and Applications Conference, 2012, pp. 233–243.

[5] G. Agosta, A. Barenghi, A. Parata, and G. Pelosi, “Automated Security Analysis of Dynamic Web Applications through Symbolic Code Execution,” in 2012 Ninth International COnference on Information Technology - New Generations, 2012, pp. 189–194.

[6] H. Al-amro and E. El-qawasmeh, “Discovering Security Vulnerabilities And Leaks In ASP . NET Websites,” in Cyber Security, Cyber Warfare and Digital Forensic (CyberSec), 2012 International Conference on, 2012, pp. 329–333.

[7] S. Van-Acker, N. Nikiforakis, L. Desmet, W. Joosen, and F. Piessens, “FlashOver : Automated Discovery of Cross-site Scripting Vulnerabilities in Rich Internet Applications,” in ASIACCS ’12: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, 2012, pp. 12–13.

[8] F. Duchene, R. Groz, S. Rawat, and J.-L. Richier, “XSS Vulnerability Detection Using Model Inference Assisted Evolutionary Fuzzing,” in 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, 2012, no. Itea 2, pp. 815–817.

 ISBN: 978-1-61208-367-4

[9] P. Bathia, B. R. Beerelli, and M. Laverdière, “Assisting Programmers Resolving Vulnerabilities in Java Web Applications,” in CCIST 2011: Communications in Computer and Information Science, vol. 133, no. 1, 2011, pp. 268–279.

[10] L. K. Shar and H. B. K. Tan, “Automated removal of cross site scripting vulnerabilities in web applications,” Inf. Softw. Technol., vol. 54, no. 5, May 2012, pp. 467–478.

[11] CWE, “CWE - CWE-79: Improper Neutralization of Input During Web Page Generation (’Cross-site Scripting') (2.5),” The MITRE Corporation. [Online]. Available: http://cwe.mitre.org/data/definitions/79.html. [retrieved: April, 2014]

[12] OWASP, “Cross-site Scripting (XSS) - OWASP,” OWASP. [Online]. Available: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS). [retrieved: March, 2014]

[13] T. Weise, Global Optimization Algorithms – Theory and Application –, 2nd Editio. 2009, p. 820.

[14] S. H. Aljahdali, A. S. Ghiduk, and M. El-Telbany, “The limitations of genetic algorithms in software testing,” ACS/IEEE Int. Conf. Comput. Syst. Appl. - AICCSA 2010, May 2010, pp. 1–7.

[15] P. R. Srivastava and T. Kim, “Application of Genetic Algorithm in Software Testing,” Intenational J. Softw. Eng. Its Appl., vol. 3, no. 4, Oct. 2009, pp. 87–96.

[16] Z. Banković, D. Stepanović, S. Bojanić, and O. Nieto-Taladriz, “Improving network security using genetic algorithm approach,” Comput. Electr. Eng., vol. 33, no. 5–6, Sep. 2007, pp. 438–451.

[17] A. B. . A. Al Islam, M. A. Azad, M. K. Alam, and M. S. Alam, “Security Attack Detection using Genetic Algorithm (GA) in Policy Based Network,” 2007 Int. Conf. Inf. Commun. Technol., Mar. 2007, pp. 341–347.

[18] A. Avancini and M. Ceccato, “Security Testing of Web Applications: A Search-Based Approach for Cross-Site Scripting Vulnerabilities,” in 2011 IEEE 11th International Working Conference on Source Code Analysis and Manipulation, 2011, pp. 85–94.

[19] A. Avancini, F. Bruno, and K. Irst, “Towards Security Testing with Taint Analysis and Genetic Algorithms,” in ICSE Workshop on Software Engineering for Secure Systems, 2010, no. Section 5, May 2010, pp. 65–71.

[20] S. V. Chandra and S. Selvakumar, “Bixsan: Browser Independent XSS Sanitizer for prevention of XSS attacks,” ACM SIGSOFT Softw. Eng. Notes, vol. 36, no. 5, Sep. 2011, pp. 1–7.

[21] Z. Tang, H. Zhu, Z. Cao, and S. Zhao, “L-WMxD: Lexical based Webmail XSS Discoverer,” in 2011 IEEE Conference on Computer Communications Workshops INFOCOM WKSHPS, 2011, pp. 976–981.

[22] PMD, “PMD: Source Code Analyzer.” [Online]. Available: http://pmd.sourceforge.net/. [retrieved: August, 2014]

[23] D. Berndt, J. Fisher, L. Johnson, J. Pinglikar, A. Watkins, and I. P. Management, “Breeding Software Test Cases with Genetic Algorithms,” in 36th Hawaii International Conference on System Sciences, Jan. 2002, pp. 1-10.

[24] B. Shuai, M. Li, H. Li, Q. Zhang, and C. Tang, “Software vulnerability detection using genetic algorithm and dynamic taint analysis,” 2013 3rd Int. Conf. Consum. Electron. Commun. Networks, Nov. 2013, pp. 589–593.

[25] OWASP, “XSS (Cross Site Scripting) Prevention Cheat Sheet,” OWASP. [Online]. Available: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet. [retrieved: April, 2014]

[26] Sourceforge Community. [Online]. Available: sourceforge.net. [retrieved: September 2014]

[27] JGAP, “JGAP: Java Genetic Algorithms Package,” 2014. [Online]. Available: http://jgap.sourceforge.net/. [retrieved: August, 2014]

 ISBN: 978-1-61208-367-4

Safety Patterns in Model-Driven Development

Timo Vepsäläinen, Seppo Kuikka
Tampere University of Technology
Dept. of Automation Science and Engineering
Tampere, Finland
{timo.vepsalainen, seppo.kuikka}@tut.fi

Abstract— **Design patterns capture named solutions to recurring challenges in development work. With an appropriate, non-restrictive tool support, design patterns could also improve the documentation value of models in model-driven development. This paper extends the design pattern modeling approach of UML Automation Profile with safety-related information and suggests the use of patterns in models to document safety aspects. The modeling concepts are tool supported. In the paper, the concepts are used for exporting safety-related documentation. The documentation can be used to guide the selection of development techniques as well as to perform consistency checks with respect to safety integrity levels that are required from modeled applications.**

Keywords-Model-Driven Development; Design Pattern; Safety.

I. INTRODUCTION

Design patterns document solutions to recurring challenges in design and development work. As a concept, design pattern was introduced in the work of Alexander [1][2] related to building architectures. In software development, design patterns began to gain popularity after the publication of the Gang of Four (GoF) patterns [3] that were targeted to object oriented software engineering. Support for the use of patterns was also developed to Unified Modeling Language (UML). Today, UML is the de-facto software modeling language. With domain specific profiles, UML is also the modeling basis of many Model-Driven Development (MDD) approaches. However, the support for design patterns in UML is still focused on describing contents of UML Classes.

The idea of MDD is to use models as the primary engineering artefacts during the development of software systems. Models describe the systems and applications from different points of view and on different abstraction levels. In MDD, the development often starts from high abstraction level models, e.g., Computation Independent Models (CIM) as in Model Driven Architecture (MDA) [4]. Model transformations are used between the models to ensure their consistency and to produce refined models based on the earlier ones. Models also document the developed systems. However, in specific application domains the required information content of documentation is governed by regulations and standards, in addition to development needs.

Safety-related systems and applications constitute such a domain. The development process of safety applications as well as solutions and techniques to be used during the process is governed by standards, e.g., IEC 61508 [5]. In addition to using standard-compliant techniques, a developer of such a system must be able to prove the compliance of it. This is where the relevant documentation is needed.

The use of MDD to safety system development has been suggested by few researchers and even less MDD has been taken to industrial practice. The reason is not that safety standards would not allow the use of MDD techniques. Instead, for example "automatic software generation" is recommended as an architecture design technique by IEC 61508 [5]. Possible explanations for the scarce use of MDD techniques in the application area are, however, the strict documentation requirements. It is possible that given the strict requirements, MDD has not been seen to offer possibilities to improve the efficiency of the development.

The purpose of this paper is to extend a design pattern modeling approach of UML Automation Profile (UML AP) [6] to safety patterns. Safety patterns are design patterns that are applicable for safety-related systems and include additional information related to safety. They can be used by exporting documentation from models of the developed systems in which the patterns are used. The documentation generation is intended to facilitate development work by: 1) supporting traceability between applicable safety solutions and their use in systems, 2) enabling verification of safety levels of patterns in comparison to required safety levels and 3) guiding the selections of techniques and solutions.

The rest of this paper is organized as follows. Section 2 reviews work related to design patterns and use of design patterns in models and model-driven development. Section 3 recapitulates the recent pattern-related work that is extended in the paper. Sections 4 and 5 present the safety-related extensions to the pattern concepts and the developed tool support, respectively. Before conclusions, Section 6 discusses the work and the relevance of safety aspects in control system development in general.

II. RELATED WORK

Support for using design patterns in UML models is in the language based on Collaboration and CollaborationUse [7] concepts that are suitable for presenting patterns inside

 ISBN: 978-1-61208-367-4

UML Classes. The concepts have been developed along the language itself from parameterized collaborations that were utilized in, e.g., [8]. In addition to the standard approach, however, many tool vendors have developed additional pattern support in a more ad hoc manner. For example, MagicDraw [9] enables the specification of model element templates and copying the templates to models to instantiate patterns. Without pointing out pattern instances, however, the information on the occurrences is endangered to vanish.

To enable precise but practical use of patterns in UML, France et al. [10] have developed a pattern modelling approach using UML. Precise specification of pattern solutions is seen to enable tool support for building solutions from pattern specifications and for verification of the presence of patterns in design. In the approach, an overall pattern specification consists of a structural pattern specification specifying the class diagram view of the solution, and a set of interaction pattern specifications that specify the interactions in the pattern solutions.

Approaches to apply and evolve design patterns to UML models have also been developed with use of model transformations [11][12][13][14] using Query/View/Transformation (QVT) and Extensible Stylesheet Language Transformations (XSLT) techniques. Detection of design patterns in models, on the other hand, has been studied for example with use difference calculation [15], graph matching [16], graph similarity scoring [17], as well as graph decomposition and graph isomorphism [18].

In the approach of the authors, the novelty is neither in the approach to transform patterns into design nor in detecting pattern instances. Instead, a starting point in the work is that uses of patterns are design decisions that should be deliberately documented by marking the patterns. On the other hand, attention is paid to the questions how the pattern markings could be used to produce documentation in general and in safety-related application development in particular.

For safety-related systems, design patterns have been specified, for example, related to redundancy. In [19], Douglass presents 4 patterns to implement redundancy or redundancy-like behavior so that a task is performed in different channels or that another computing channel is used to observe the behavior of the main channel. Also IEC 61508 [5] in the 6th part of it presents several M out of N solutions in which the idea is to perform a calculation redundantly and to use voting to acquire a reliable result for it.

In the tables of recommended techniques and measures for software architecture design (annex A), IEC 61508 [5] also refers to a wide range of solutions that already have corresponding patterns in pattern literature. For example, the standards suggest the use of (different kinds of) redundancy [19], backward recovery (from faults) [20][21] and cyclic program execution [19]. Another example on use of patterns in the domain is related to documenting recurring arguments of safety cases in order to systematically collect and gain benefit from arguments of previous projects [22].

MDD of safety systems has been studied in the DECOS project [23] that is targeted to development of both critical and non-critical functions of embedded control systems. In the approach, the preferred means for specifying application functionality is Safety-Critical Application Development Environment (SCADE) which is based on formally defined data flow notation and enables simulation at model level and code generation.

UML based modelling and development of safety applications has also been facilitated with UML profiling techniques. In [24] the approach is based on extracting key concepts of a safety standard, RTCA DO-178B, to stereotypes with which it is possible for software developers to include safety-related concepts and properties in models. It can be assumed that such models suit well also for the purpose of producing documentation. However, we regard the work presented in this paper as an important complement to the approach. Whereas UML stereotypes are applied to single modelling elements, with patterns it is possible to link several elements in designs to patterns and roles of them. This is needed in order to characterize how a number of elements are used together to perform a task.

III. NEED FOR PATTERNS IN MDD

The key concept of MDD is to shift the development efforts from written documents to models that are used throughout the development process. For special purposes, e.g., safety system development, it could be possible to maintain separate documents. However, that would require additional work and could significantly reduce the potential to benefit from MDD. In a sense, it would also be against the central idea in MDD. A more appropriate approach would be to include the documentation in the models, in the first place.

A possible challenge in this objective is that models, in general, tend to be more applicable for representing solutions than rationale behind them. For example, many of the basic concepts of UML are similar to concepts of object oriented programming languages. UML models can be well used to answer the question how to implement, e.g., a class or a program. In the MDD context, it is even possible to generate code from models to avoid the manual programming work. However, information on why something has been designed in the way it has, is often missing. This information could be crucially important for, e.g., quality assurance and maintenance purposes.

Design patterns are a possible solution to improve the situation. Patterns document named, proven solutions that are well-known among developers and suited for solving recurring challenges and tasks. They are structured so that they consist of named parts that have responsibilities in the solutions. The solutions that patterns include may have crucial advantages. The use of design patterns and pattern instances in MDD and models could thus increase the value of models significantly. Patterns could 1) indicate the use of standard solutions in systems and specifications, 2) mark potential challenges (that are treated with the patterns), 3) make design more understandable (because of the use of the known solutions) and 4) clarify the roles of model elements in design, just to name a few benefits. In specific application areas, e.g., safety system development, the use of patters could even automate tasks and checks that are currently performed manually.

 ISBN: 978-1-61208-367-4

A. Design Patterns in UML

In UML, pattern definitions and pattern instances are defined with the Collaboration and CollaborationUse concepts of the language, respectively. Similarly to the Class concept, Collaboration extends the StructuredClassifier and BehavioredClassifier concepts. A pattern definition is in the language a set of cooperating participants that are Properties of a Collaboration. In a similar manner Properties can be owned by Classes. The features that are required from the participants are defined by the Classifiers that are used as types of the Properties. Graphically Collaborations can be presented in composite structure diagrams in which participants of a pattern are connected with Connectors.

A CollaborationUse represents an application of a pattern to another Classifier (Class). The CollaborationUse must be owned by the Class to the contents of which it (the pattern) is applied. Properties of the applying Class can be bound to the roles of the Collaboration with Dependencies. The entities playing the roles must be owned by the same Class instance that owns the CollaborationUse. In short, with the UML pattern concepts, patterns are seen to describe contents of Classifiers.

Pattern literature of today, however, is not restricted to contents of UML Classifiers only. For example, many well-known patterns such as the Layers pattern [25] (and many other architectural patterns) are intended to clarify the division of systems to, e.g., Components or Packages. However, marking the occurrence of such patterns may not be possible with the UML concepts. This is because Packages are not Properties or necessarily owned by Classes. With application domain specific extensions, the support for patterns in UML becomes even more constraining. In order to benefit from the use of patterns in MDD, a new approach to define and mark patterns in models is required. The approach should restrict neither the types of elements that play roles in patterns nor the types of elements to contents of which patterns can be applied.

B. The New Pattern Approach

The developed pattern modelling approach [6] uses a set of concepts that have been developed for defining patterns and marking pattern instances in models. In the approach, pattern instances are not owned by Classes but Packages that are used in models in any case. The elements playing pattern specific roles in pattern instances can be any direct or indirect contents of the Packages and instances of any metaclass, instead of Properties only. Pattern definitions include textual properties that are essential information content in patterns. Lastly, the element roles in pattern definitions are separated from the template elements that are used in automating the application of patterns.

The approach is tool-supported including functions for instantiating patterns, exporting statistics and traceability information related to the use of patterns as well as for visualizing patterns in diagrams [6]. Patterns are instantiated to models with the use of a wizard that performs pattern specific modifications to the models, according to user selections. Markings of pattern instances are also created automatically by the wizard.

Statistics and traceability information on patterns can be exported to MS Excel files. Statistics include lists of design patterns that are used in a model including the number of instances for each pattern. Patterns are traced to Packages with traceability matrices to indicate the patterns that are used in each Package and vice versa. Visualizing patterns in diagrams utilizes the Collaboration notation of UML and presents pattern instances with dotted ellipses. Model elements that play pattern specific roles in the instances are connected to the ellipses with dotted lines. The tool support for the use of patterns can be used in any UML, Systems Modeling Language (SysML) or UML Automation Profile (AP) models and diagrams in UML AP research tool [26].

IV. SAFETY PATTERN METAMODEL

With extensions to safety aspects, the purpose has been to experiment how design patterns could specifically support documentation of safety applications. Most importantly, the extensions to the pattern modeling concepts, see Figure 1, include a specific SafetyPattern. SafetyPatterns are design patterns that have been identified to be related to safety. To distinguish the concepts that are used for defining patterns from those used to mark pattern instances, the Figure has been divided to two parts. The new (in comparison to [6]) concepts are in the Figure high-lighted with grey color.

A SafetyPattern is, thus, a design pattern that has been identified to be related to safety and that may have recommendations for applications of different *safety levels*. With safety systems, we refer to systems that perform *safety functions* the correct operation of which is required to ensure the safety of a controlled process. The safety levels in the metamodel correspond to the 4 Safety Integrity Levels (SILs) in IEC 61508 [5]. In general, a SIL determines the probability of correct functioning of a safety function, SIL1 being the lowest and SIL4 being the highest level. For traditional, e.g., electrical safety systems it is possible to determine SILs statistically. However, due to the systematic (vs. random) nature of software faults, the statistics approach cannot be applied to software. For new software components there would not even be statistics available. In IEC 61508, this problem is solved by focusing on development techniques and solutions the use of which are documented. For each SIL and for each development phase, the standard specifies a set of techniques that can be highly recommended (HR), recommended (R) or non-recommended (NR) or with non-specified recommendation (NS). The alternatives in the Recommendation (enumeration) in the metamodel correspond to these alternatives.

The purpose of the SafetyCatalogue concept is to collect together (from various pattern sources) related SafetyPatterns. Catalogues contain patterns that should be used together and to which sets of patterns that are used in models can be compared. Patterns in a catalogue can be related to, e.g., a phase in development or a specific purpose. For example, IEC 61508 [5] includes lists of techniques to be used during specific software development phases. For software architecture design, for instance, the standard mentions 27 techniques and/or measures, some of which are non-recommended or alternatives to each other.

 ISBN: 978-1-61208-367-4

Relations between Patterns can be modeled with the PatternRelation concept that has been extended with a Specialization relation. The background of the new (specialization) relation is an observation that many solutions (such as redundancy) that are recommended by safety standards actually have families of related, specialized pattern versions in pattern literature. With the Specialization relation, the purpose is to enable the use of general SafetyPatterns in SafetyCatalogues but in such a way that patterns specializing the general patterns can be considered as their alternatives.

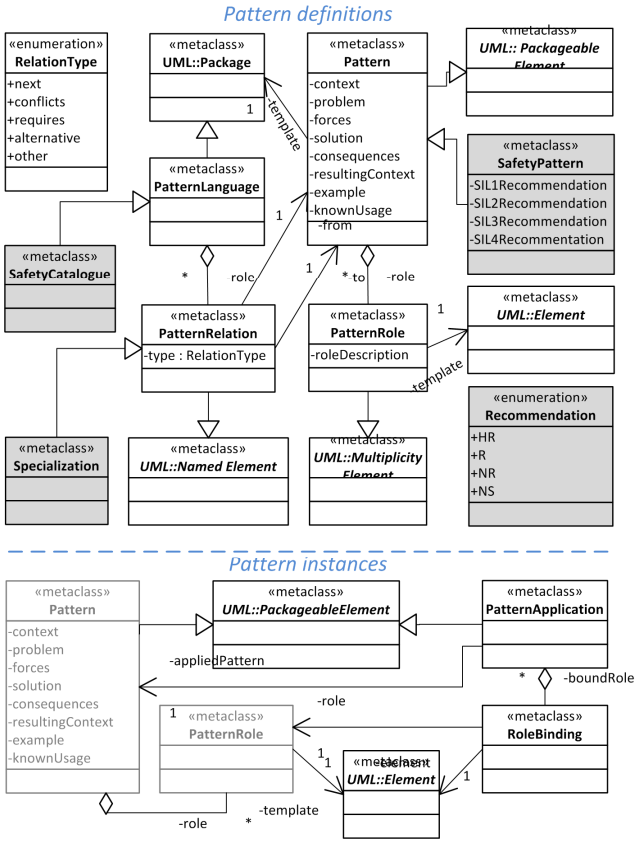


Figure 1. The new concepts for defining and using safety patterns.

The modeling concepts have been implemented to UML AP Tool [26]. With the implementation, the purpose has been to demonstrate how the concepts can be used to generate safety-related documentation. The implementation of the concepts uses Eclipse Modeling Framework (EMF) as a meta-modeling framework, with which the new concepts have been defined by extending the existing UML AP modeling concepts. The developed documentation generation extends the work presented in [6] and [27] that already addresses, e.g., traceability of requirements.

V. FOR GUIDANCE AND DOCUMENTATION

In this Section, we present three example documentation sheets. The generation of the sheets has been automated with use of the concepts. In addition to discussing how the sheets can be used, the following sub-Sections will briefly describe how the sheets are compiled from models.

The first of the sheets to be presented was created based on a SafetyCatalogue that had been defined to correspond to recommendations of IEC 61508 to software architecture design. The latter two example sheets compare a set of SafetyPatterns that is used in an example model to another SafetyCatalogue. The generation of the sheets relies on patterns that have been identified to be related to safety and that include recommendations for the different levels of safety.

A. *Safety Catalogue Sheet*

The purpose of the Safety Catalogue sheet is to enable illustrating SafetyCatalogues in a tabular form that is similar to the form of recommendation tables of IEC 61508 [5] (annex A of part 3 of the standard). On one hand, the sheet has been developed to facilitate the development of SafetyCatalogues, including checks of their conformance to standards. The tabular presentation can be used also during development to look for possible patterns or solutions that should be applied during specific design phases.

In addition to recommendations of safety standards, the sheet enables illustrating custom catalogues of SafetyPatterns for which there may not be standard recommendations. Nevertheless, such patterns may provide solutions to similar problems and be alternatives to each other. On the other hand, it may be meaningful to represent in which order such patterns should be applied so that composing pattern catalogues with next and alternative relations can be useful.

The Safety Catalogue sheet is compiled as follows. PatternApplications of an exported model are iterated through to find all SafetyPatterns that are used in the model. The SafetyPatterns are iterated through to find the SafetyCatalogues in which they appear. The list of the catalogues is provided to the user of the tool. The selected catalogues are printed to separate tables starting from their first patterns that are assigned number 1 in the tables. Next and alternative SafetyPatterns can be found with use of the PatternRelations. Alternatives are in the tables assigned same numbers but different letters, to indicate them being alternatives to each other. Recommendations of the SafetyPatterns to SILs are printed to the tables.

Safety Catalogue: "IEC 61508 Architecture Design"

#	Pattern	SIL 1	SIL 2	SIL 3	SIL 4
1	Fault detection	NS	R	HR	HR
2	Error detecting codes	R	R	R	HR
3a	Failure assertion programming	R	R	R	HR
3b	Diverse monitor techniques (with independence)	NS	R	R	NS
3c	Diverse monitor techniques (with separation)	NS	R	R	HR
3d	Diverse redundancy	NS	NS	NS	R
3e	Functionally diverse redundancy	NS	NS	R	HR
3f	Backward recovery	R	R	NS	NR
3g	Stateless software design	NS	NS	R	HR
4a	Re-try fault recovery mechanism	R	R	NS	NS
4b	Graceful degradation	R	R	HR	HR
5	Artificial intelligence - fault correction	NS	NR	NR	NR
6	Dynamic reconfiguration	NS	NR	NR	NR
7	Modular approach	HR	HR	HR	HR
8	Use of trusted/verified software elements (if available)	R	HR	HR	HR
...					

Figure 2. An example generated Safety Catalogue sheet.

 ISBN: 978-1-61208-367-4

An example Safety Catalogue sheet can be seen in Figure 2 that presents a part of a printout of a catalogue of techniques or measures that IEC 61508 recommends for software architecture design. In the table, patterns can be highly recommended (HR), recommended (R) or non-recommended (NR) or with non-specified recommendation (NS). To avoid repeating a table of the standard, the table includes only 15 techniques that have been modeled as patterns. By looking at the table, however, it also becomes clear that pattern literature already includes specialized versions of many of the techniques, for example to implement redundancy [19].

B. *Safety Catalogue Conformability Sheet*

Whereas the purpose of the Safety Catalogue sheet is to enable presenting catalogues of SafetyPatterns, the purpose of Safety Catalogue Conformability sheets is to present how a set of SafetyPatterns (that are used in a model) conforms to a SafetyCatalogue. Similarly to the previous sheet, the conformability sheet serves both the guidance and documentation purposes. In addition, the table presents to which SILs the set of SafetyPatterns would be applicable.

The sheet is compiled as follows. In a similar manner than in the case of the previous sheet, the SafetyCatalogues related to the model are collected to a list from which the user may select the desired ones. General structure of the sheet is similar to the previous sheet. However, the SafetyPatterns of the catalogue that are used in the model are indicated with light grey color. In addition, the table presents whether the set of (used) patterns is compatible with each SIL. Compatibility of the used patterns is illustrated with green color and incompatibility with red color. Incompatibility can result from both using a non-recommended pattern or not-using a recommended (or highly recommended) technique or any of its alternatives.

The last two rows of the table also present the numbers of patterns (excluding alternatives) that would be recommended for each SIL and how many of them have been actually applied. As such, the table also answers the question how many techniques (more) should be applied in order to conform to the catalogue for each SIL.

Conformability to "IEC 61508 Software safety requirements specification"

#	Pattern	SIL 1	SIL 2	SIL 3	SIL 4
1a	Semi-formal methods	R	R	HR	HR
1b	Formal methods	NS	R	R	HR
2	Forward traceability between the system safety requirements and the software safety requirements	R	R	HR	HR
3	Backward traceability between the safety requirements and the perceived safety needs	R	R	HR	HR
4	Computer-aided specification tools to support appropriate techniques/measures above	R	R	HR	HR
	Pattern usage:	4/4	4/4	4/4	4/4
	Pattern usage (%):	100.0	100.0	100.0	100.0

Figure 3. An example generated Safety Catalogue Conformability sheet.

An example Safety Catalogue Conformability sheet can be found in Figure 3. It presents the conformability of SafetyPatterns used in an example model to the software safety requirement specification techniques of IEC 61508 [5] that have been modeled as a SafetyCatalogue. According to the table (grey highlighting), it can be seen that a semi-formal modeling technique has been used, the software safety requirements specification supports both backward and forward traceability and that computer-aided specification tools have been used. The table also illustrates (with green color) that these choices are applicable to all SILs. In addition to the techniques used, it is not necessary to use any other technique (for requirements specification).

C. *Safety Pattern Traceability Sheet*

While patterns can have recommendations for different levels of safety, it is also possible to check their conformance to safety levels required from the safety functions. The purpose of the safety pattern traceability sheet is to trace safety requirements (of UML AP) to Packages that contain implementing design elements for the requirements and to SafetyPatterns that are used in the Packages. In addition to traceability, the table presents the safety levels (SIL) related to the requirements, Packages as well as recommendations of the Patterns for each level. Similarly to the previous sheet, the use of recommended or highly recommended patterns is indicated with green color whereas the use of non-recommended patterns is warned with red color.

The sheet is compiled as follows. Safety-related (UML AP) requirements and their respective safety integrity levels are collected to a list. The Packages that contain implementing design elements for the requirements are identified based on TraceRelations (of UML AP). The SafetyPatterns, instances of which can be found from the Packages, are identified based on PatternApplications. The traceability table is printed. In the table, traceability between a requirement and a Package is presented with an arrow (↘). SILs for the Packages are determined by finding the highest SILs from the requirements that are traced to the Packages. Traceability between a Package and a SafetyPattern used in the Package is, again, presented with the arrow symbol.

Traceability: Requirements --> Packages --> (Safety) Patterns

REQUIREMENT:							
P100 Protection (SIL1)	↘						
P100IR (SIL1)		↘					
PACKAGE:	Software Safety Requirements (SIL1)	ControlStructures (SIL1)	PATTERN:	SIL 1	SIL 2	SIL 3	SIL 4
		↘	Automatic software generation	R	R	R	R
	↘		Semi-formal methods	R	R	HR	HR
	↘		Backward traceability between the safety requirements and the perceived safety needs	R	R	HR	HR
	↘		Computer-aided specification tools to support appropriate techniques/measures above	R	R	HR	HR
	↘		Forward traceability between the system safety requirements and the software safety requirements	R	R	HR	HR

Figure 4. An example generated Safety Pattern Traceability sheet.

An example Safety Pattern Traceability sheet can be found in Figure 4. According to the table, it can be seen that the example model contains 2 requirements of safety level SIL1: P100 protection and P100IR. The former one (a general safety function requirement) is traced to "Software Safety Requirements" Package and the latter one to "ControlStructures" Package. SILs required from the Packages (their contents) come from the requirements, both

 ISBN: 978-1-61208-367-4

being SIL1. Moreover, the sheet presents that the use of "Automatic software generation" has been marked in ControlStructures Package and Semi-formal methods, backward traceability, forward traceability as well as computer aided specification tool in the Software Safety Requirements Package. According to the table (color coding), the techniques are recommended for the safety integrity level (SIL1) required from the Packages.

VI. DISCUSSION

This paper has presented an approach to extend the information content of design pattern concepts of UML AP with safety aspects. The new concepts enable specifying the applicability of SafetyPatterns, i.e., design patterns of safety systems, to applications of different safety integrity levels. In addition, SafetyPatterns can be collected to SafetyCatalogues with which it is possible to model both recommendations of safety standards and custom catalogues of SafetyPatterns.

To illustrate the use of the concepts, the paper has presented 3 example documentation sheets. The sheets were generated automatically based on a library model containing two SafetyCatalogues and a model utilizing the patterns of the catalogues. The first of the sheets presented one of the catalogues. The other two sheets presented compliance of a model (of a developed systems) to the other catalogue. The new information content of SafetyPatterns was in the sheets used for automating identification of safety-related patterns and consistency checks with respect to safety levels. The sheets, thus, documented rather the developed systems than SafetyPatterns themselves. In the developed metamodel, SafetyPatterns share most of their information content with the design pattern modeling concepts that are used in [6].

The authors believe that the possibility to export documentation from models is a future research topic within MDD research. Moreover, it could improve the applicability of the MDD techniques to safety system development. This is because safety applications cannot be used in practice without appropriate documentation. Without automated support for producing documentation, it would have to be produced manually. On the other hand, by automating even part of the work, it would be possible to obtain additional, MDD specific benefits in the application area.

When developing safety applications with MDD techniques, the development process should be supported. A tool should assist developers by pointing out the issues that need to be addressed, by presenting the alternatives (when appropriate) and by documenting the decisions for later use. For example, the supported process could start from modeled requirements that determine the required integrity levels. A developer could select a SafetyCatalogue to be used to guide, e.g., architecture design. Based on the selection and required integrity levels, the tool could suggest patterns to be used. In practice, this scenario could be supported with only a small modification to the Safety Catalogue sheet, by hiding inappropriate patterns based on required integrity levels.

Work that aims for guiding development work in MDD has been previously carried out by the authors also based on use of an Architecture Knowledge Management (AKM) platform [28]. Use of an external tool, however, may lead to redundant information. On the other hand, it is believed that documentation and guidance support should be available for both architectural and detailed design levels. Thus, it is feasible to integrate the required support in one tool, which is used throughout the MDD process.

A challenge in developing guidance for MDD is that development processes, techniques and solutions vary between companies and between controlled processes. The approach presented in this paper could improve the situation. Documentation sheets can be developed to support various purposes and processes, not only the ones presented in this article. In addition, by using, e.g., the SafetyCatalogue concept, the generated sheets and their contents are also dependent of the catalogues to the contents of which the models are compared. Thus, to support another kind of a development process or other techniques, one could specify other catalogues to which the models would be compared.

The authors regard safety aspects important for also basic control systems that are not critical. Safety is an issue that should be taken into account in development of any control system. Safety standards state their recommendations on techniques, measures and solutions based on evidence on their usefulness. It is likely that adopting selected techniques and measures from safety system development, e.g., traceability could also improve the quality of basic control systems. This could in turn improve the productivity of the controlled processes at least in application domains in which the development processes are not strictly governed.

On the other hand, considering selected aspects of safety standards in development of basic control systems could shorten the gap between the systems. Safety systems and basic control systems are currently not only separated from each other but also developed with different development processes and tools and often by different teams. It is possible that professionals are not even aware of the practices in the other teams. Because the development of safety systems is regulated by authorities, the only possibility to shorten the gap would be to adopt suitable practices of safety system development to basic control system development.

VII. CONCLUSIONS

Design patterns document solutions and capture expert knowledge to recurring challenges in design and development work. On one hand, design patterns support the re-use of design by preserving named, proven solutions to recurring challenges. However, they can also increase the documentation value of models that usually tend to present design solutions rather than rationale behind the solutions. With use of patterns, designs become easier to understand and the roles of design elements clear for possible third parties that use the documentation. Especially the use of patterns could benefit MDD in which the idea is to use models for both development and documentation purposes.

In this paper, a set of pattern modeling concepts was presented that enable increasing the information content of design patterns with applicability to safety integrity levels. The new concepts enable constructing catalogues of safety-related patterns with which it is possible to model

 ISBN: 978-1-61208-367-4

recommendations of safety standards. Automated functions for generating documentation sheets enable the use of the concepts for producing documentation. In addition to presenting which patterns are used in a model, the sheets present whether the models comply with the catalogues, e.g., recommendations of safety standards. The sheets can be used also during development as guidance to present the standard-compliant selections that still have to be addressed.

Ability to use models as documentation or to produce documentation from models to a suitable form is a possible key for industrial acceptance of MDD techniques in safety system development. Without automated support, the documentation would have to be produced manually. This could significantly reduce the potential to benefit from MDD. However, with documentation support, MDD would provide another means to benefit from the use of models.

When developing safety applications with MDD techniques, the development process should be supported and guided in a flexible manner. Instead of only predefined forms and checks, the presented documentation tables are compiled with use of modelled SafetyCatalogues to which models are compared. As such, the suggestions that the tool can be considered to provide are also dependent on the modelled catalogues. Tailoring the approach for different application domains or development practices could thus be possible to achieve with changes to the catalogues. While acknowledging that the development concepts still require further development, the authors regard this kind of flexibility as an important feature in MDD tool support.

REFERENCES

[1] C. Alexander, S. Ishikawa, and M. Silverstein, "A pattern language: towns, buildings, construction", Oxford University Press, 1977.

[2] C. Alexander, "The timeless way of building", Oxford University Press, 1979.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design patterns: Elements of reusable object-oriented software", Addison-Wesley, 1994.

[4] OMG, "Model Driven Architecture (MDA) Guide", Object Management Group, 2003.

[5] IEC, "61508 functional safety of electrical/electronic/programmable electronic safety-related systems – Part 3: Software requirements", International Electrotechnical Commission, 2010.

[6] T. Vepsäläinen and S. Kuikka, "Design pattern support for model-driven development", in 9th International Conference on Software Engineering and Applications, 2014. (in press)

[7] OMG, "Unified Modeling Language Specification 2.4.1: SuperStructure", Object Management Group, 2011.

[8] G. Sunyé, A. Le Guennec, and J. Jézéquel, "Design patterns application in UML", in Proc. of 14th European Conference on Object-Oriented Programming, 2000, pp. 44-62.

[9] No Magic, Inc. MagicDraw, 2014. Available: http://www.nomagic.com/products/magicdraw.html [retrieved: 07, 2014]

[10] R. B. France, D. Kim, S. Ghosh, and E. Song, "A UML-based pattern specification technique", IEEE Transactions On Software Engineering, vol. 30, pp. 193-206, 2004.

[11] J. Dong, Y. Sheng, and K. Zhang, "A model transformation approach for design pattern evolutions", in Proc. of 13th Annual IEEE International Symposium and Workshop On Engineering of Computer Based Systems, March 2006, pp. 80-92.

[12] P. Kajsa and L. Majtás, "Design patterns instantiation based on semantics and model transformations", in SOFSEM 2010: Theory and Practice of Computer Science, Springer, 2010, pp. 540-551.

[13] W. Xue-Bin, W. Quan-Yuan, W. Huai-Min, and S. Dian-Xi, "Research and implementation of design pattern-oriented model transformation", in 2nd International Multi-Conference on Computing in the Global Information Technology, 2007.

[14] J. Dong and S. Yang, "QVT based model transformation for design pattern evolutions", in Proc. of 10th IASTED International Conference on Internet and Multimedia Systems and Applications, 2006, pp 16-22.

[15] S. Wenzel and U. Kelter, "Model-driven design pattern detection using difference calculation", in Proc. of 1st International Workshop on Pattern Detection for Reverse Engineering, October 2006.

[16] M. L. Bernardi, M. Cimitile, and G. A. Di Lucca, "A model-driven graph-matching approach for design pattern detection", in Proc. of 20th IEEE Working Conference on Reverse Engineering, 2013, pp. 172-181.

[17] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. T. Halkidis, "Design pattern detection using similarity scoring", IEEE Transactions On Software Engineering, vol. 32, pp. 896-909, 2006.

[18] A. Pande, M. Gupta, and A. K. Tripathi, "A new approach for detecting design patterns by graph decomposition and graph isomorphism", in Proc. of 3rd International Conference on Contemporary Computing, Springer, 2010, pp. 108-119.

[19] B. P. Douglass, Real-Time UML: Developing Efficient Objects for Embedded Systems. Addison-Wesley, 1998.

[20] R. Hanmer, Patterns for Fault Tolerant Software. John Wiley & Sons, 2013.

[21] T. Saridakis, "Design patterns for checkpoint-based rollback recovery," in Proc. of 10th Conference on Pattern Languages of Programs (PLoP), Spetember 2003.

[22] T. P. Kelly and J. A. McDermid, "Safety case construction and reuse using patterns. in Proc. of 16th International Conference on Computer Safety and Reliability, Springer, 1997, pp. 55-69.

[23] W. Herzner et al., "Model-based development of distributed embedded real-time systems with the decos tool-chain," in Proc. of 2007 SAE AeroTech Congress & Exhibition, 2007.

[24] G. Zoughbi, L. Briand, and Y. Labiche, "Modeling safety and airworthiness (RTCA DO-178B) information: conceptual model and UML profile", Software & Systems Modeling, vol. 10, pp. 337-367, 2011.

[25] F. Buschmann, R. Meunier, H. Rohnert, P Sommerlad, and M. Stal, "Pattern Oriented Software Architecture: A System of Patterns". John Wiley & Sons, 1996.

[26] T. Vepsäläinen, D. Hästbacka, and S. Kuikka, "Tool support for the UML automation profile - for domain-specific software development in manufacturing", in Proc. of 3rd International Conference on Software Engineering Advances, October 2008, pp. 43-50.

[27] T. Vepsäläinen and S. Kuikka, "Towards model-based development of safety-related control applications", in the 16th IEEE International Conceference on Emerging Technologies & Factory Automation, September 2011.

[28] T. Vepsäläinen, S. Kuikka, and V. Eloranta, "Software architecture knowledge management for safety systems", in the 17th IEEE International Conceference on Emerging Technologies & Factory Automation, September 2012.

 ISBN: 978-1-61208-367-4

Test Data Generation Based on GUI: A Systematic Mapping

Rodrigo Funabashi Jorge
Faculdade de Computação
Universidade Federal de Mato Grosso do Sul
Campo Grande, MS, Brazil
Email: rodrigo.funabashi@ufms.br

Márcio Eduardo Delamaro
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo
São Carlos, SP, Brazil
Email: delamaro@icmc.usp.br

Celso Gonçalves Camilo Junior
Instituto de Informática
Universidade Federal de Goiás
Goiânia, GO, Brazil
Email: celso@inf.ufg.br

Auri Marcelo Rizzo Vincenzi
Instituto de Informática
Universidade Federal de Goiás
Goiânia, GO, Brazil
Email: auri@inf.ufg.br

***Abstract*—For the general case, the complete automation of test data generation is an undecidable problem, and many researches employ meta-heuristics trying to find a reasonable partial solution. System testing performed via Graphical User Interface (GUI) imposes extra challenge for automation due to hundreds and often thousands of possibilities of events that can be generated. This work presents a study based on systematic mapping aiming at identifying the state of the art and the state of the practice on the automation of system testing carried out via GUI. We employed the traditional protocol of mapping study to support the data collection. The work was carried out from 6th February 2012 to 1st May 2013 resulting in the selection of 39 out of 598 primary studies obtained with the application of the search strings. Some of these works used, besides functional testing criteria, structural testing criteria to guide meta-heuristics. In relation to meta-heuristics, the distribution of work was more uniform, with a slight majority using Genetic Algorithms for test data generation. There are few research groups working on this subject. One particular author is responsible for authoring more than 30% of the selected primary studies and can be considered a reference in the generation of test data from GUI. Some research problems identified are 1) the difficult to represent all the possible GUI interactions without cause state explosion, 2) the need to evaluate the techniques on large software products, and 3) the complexity to automate the representation of the GUI interactions by reducing the number of infeasible sequences of actions.**

Keywords–Systematic Mapping Study; System Testing; Testing through GUI; Automated Test Data Generation

I. INTRODUCTION

Modern software systems include various components that interact with each other to perform tasks. The correct behavior of these components is often verified by means of unit tests, integration, system and acceptance. In unit testing, the goal is to identify logic and implementation faults on each software unit. Integration testing is a systematic technique to integrate the software units in order to identify faults in the communication interface between them. The system test is performed after the system integration and aims to ensure that the software meets all functional, behaviour and performance requirements described in the specification. Finally, acceptance testing aims to verify whether the developed product meets the requirements specified by users [1]. Many of today's applications have a special component in the form of a Graphical User Interface (GUI). The GUI is composed by a set of control elements *widgets*, such as buttons, menu items and text boxes. The graphical interface is often the only piece of software which the user interacts. This way, it is necessary to test this interface in order to ensure product quality through the creation of test data in the form of input sequences. An input for a GUI application is a sequence of actions, such as clicking a button control or dragging and dropping GUI elements. It is observed that, in general, this type of test is performed during system and acceptance testing.

To generate a good set of test data, test designers should make sure that this set covers all the features of the system and, in the context of GUI also has exercised all possibilities of interface events. However, the difficulty in performing this task is twofold: to manage the domain size and to sequence the actions.

Within the first problem, unlike a system with a command line interface, a graphical user interface has many operations that need to be tested. A relatively small program, such as Microsoft WordPad has 325 possible GUI operations [2]. Therefore, the number of operations can easily be an order of extremely high magnitude for more complex programs [3].

Regarding the second issue, some functionality of the system can only be performed following a complex sequence of GUI events. For example, to open a file the user must enter the File menu and select the Open operation and then use a dialog box to specify the file name and complete the operation. Obviously, increasing the number of possible operations increases the sequencing problem exponentially, making it difficult to create test data manually.

Due to the difficulties related to the generation of test data which run via GUI, this work focused on the collection of primary studies in this context by applying a systematic mapping to verify what is available in the literature on this subject and which gaps can still be explored on further researches. In Section II, an explanation about GUI test, its features, limitations and related issues are presented. In Section III describes the planning, conduction, and analysis of the application of the systematic mapping. The conclusion is presented in Section IV.

II. GUI TESTING

Graphical user interface is a type of interface that allows the user to interact with digital devices through graphical elements such as icons and other visual indicators, as opposed to the

 ISBN: 978-1-61208-367-4

command line interface. The interaction is usually done via mouse or keyboard, on which the user is able to select symbols and manipulate them in order to get some practical result. These symbols are referred to *widgets* and are grouped into *kits*.

In our context, the testing via GUI means to perform a system or acceptance testing of a particular product to ensure it meets its specifications. This is normally done by using a range of test data.

To generate a good set of test data, designers should check that this set covers all the features of the system and, in the context of GUI also has exercised all possibilities of interface events. However, there are some open problems related to GUI testing [3]:

1) the huge amount of possible sequences from each state, i.e., in every state there are many alternative actions leading to an exceptionally large search domain. Furthermore, it is computationally expensive to generate and evaluate sequences, since the software needs to be started and all actions need to be performed in sequence. This requires efficient algorithms to explore the search space in an intelligent way to find optimal test sequences;
2) related to the generation of inputs for applications that explore button clicks and drag and drop operations components:
    a) map the GUI to determine the visible *widgets* and their properties. For example, the position of the components such as buttons and menu items;
    b) derive a set of permitted actions at each stage of implementation. For example, a visible button may be disabled and could not be pressed; and
    c) perform and record the test sequence making it possible to repeat (play) it again later.

To deal with the nature of the problems mentioned, the Artificial Intelligence is very applied, since these are optimization problems. Thus, the research area called *Search-based Software Engineering (SBSE)* has emerged, which deals with the application of mathematical optimization techniques to solve complex problems in the field of Software Engineering. According to the Software Engineering by Automated Search (SEBASE) website [4], that maintains a updated database about SBSE, 52% of the publications on SBSE focus on testing and debugging. This is due to the high cost of implementing these activities, which in general can spend 50% of the development cost [1]. Given this scenario a subarea of SBSE called *Search-Based Software Testing (SBST)* was created, focusing on the application of mathematical optimization techniques in solving problems in the context of testing activity. Therefore, the challenge is to automate the testing process as much as possible, and the generation of test data is, of course, an essential part of automation. Also according to the site SEBASE, only 6% of works in this area belong to Brazil.

III. Application of Systematic Mapping

Systematic mapping is a type of literature review [5], in which it is conducted a broader review of primary studies to identify researches evidences and gaps, directing the focus of future systematic reviews, which tries to answer more specific research questions [5]. The systematic mapping study provides an overview of a research area, the amount, the type of research conducted, the results are available, in addition to the frequency of publications over time to identify trends [6]. There are many reasons for conducting a systematic mapping, the most common being [7]:

- to summarize the existing evidence regarding treatment or technology, for example, to summarize empirical evidence of the benefits and limitations of a specific method;
- to identify gaps in current research in order to suggest future areas of research; and
- to provide an overview/subsidy for advancing knowledge in new areas of research.

However, the mapping can also be used to examine the extent to which the empirical evidence supports/contradicts theoretical assumptions, or even to aid in the generation of new hypotheses [5], being composed of the following steps: planning, conducting and reporting [5].

A. Planning

The planning of this systematic mapping, which describes the protocol that was established, was carried out from the adaptation of the protocol model presented by Petersen et. al. [6], that specifies the following elements [5]: research questions, search strategy and implementation, criteria for inclusion and exclusion, and data extraction and synthesis methods.

Research questions define the scope of the mapping. They guide the development of the remainder of the study and should be set according to the motivations of the study [6]. The research questions (RQ) were elaborated in order to find primary studies to understand and summarize evidences on the adoption of techniques for automatic generation of test data from GUI:

- $\mathbf{RQ}_1$: Do the techniques employed in GUI testing intend to cover a specific test criterion?
- $\mathbf{RQ}_{1.1}$: What is the test criterion?
- $\mathbf{RQ}_2$: What are the heuristics, techniques, algorithms or strategies used for automatic generation of test data from GUI?
- $\mathbf{RQ}_3$: Do the techniques for automatic generation of GUI test data require a data model that abstracts the GUI to perform the test generation?
- $\mathbf{RQ}_{3.1}$: If need, is the model generated automatically or manually?
- $\mathbf{RQ}_4$: What are the available tools and how do they support the automatic generation of test data from the GUI?
- $\mathbf{RQ}_5$: In what domain are automatically generating test data based on GUI applied?

Systematic mapping is a kind of secondary study in Software Engineering [8], identifying primary studies from several sources (databases). These sources can be classified into two main categories [8]: index engines and sites of editors. The index engines work with several publishers publications. One

 ISBN: 978-1-61208-367-4

can cite SCOPUS as an example of index engines. The sites of editors refer to databases of online literature supplied by the editors to facilitate the recovery of the published literature. A popular site of editors in computing is the IEEE. However, as is the case of the ACM, some of these sources fall into two categories. The bases chosen in this study are ACM, IEEE, *Science Direct*, and SCOPUS, considered to be relatively efficient in conducting systematic reviews and mappings in the context of Software Engineering [9].

To build the search string, key concepts that wish to investigate were selected. From this, the synonyms, related terms and acronyms were identified. Related to the concept "Graphical User Interface" were *graphical user interface*, GUI e *web application*. For the concept "Automatic Test Data Generation" were *test data generation*, *test-data generation*, *generating test data*, *generate test data*, *automated testing*, *automation testing*, and *automation test*.

Based on the above key concepts, the default search string was built using the Boolean AND/OR connectors:

("graphical user interface" OR "GUI" OR "web application") AND ("test data generation" OR "test-data generation" OR "generating test data" OR "generate test data" OR "automated testing" OR "automation testing" OR "automation test")

A total of 10 articles were selected [10]–[19]. These articles have provided evidence that this search string is adequate, since all these items were returned after the application of search string in their respective bases.

To determine the relevancy of given primary study it must satisfy any Inclusion Criteria (IC) on the other hand it will be excluded by any Exclusion Criteria (EC). Our inclusion criteria are:

- **IC_1:** The study presents a case study or experience report using techniques for generating test data from GUI;
- **IC_2:** The study presents an investigation of the technical features to generate test data from GUI;
- **IC_3:** The study proposes methods for evaluating techniques for generating test data from GUI;
- **IC_4:** The study presents tools that use techniques to generate test data from GUI.

Primary studies considering different domains or presenting ideas in a vague way were excluded. To classify these studies the following exclusion criteria were identified:

- **EC_1:** The work is not related to any of the research questions;
- **EC_2:** The work was selected by another search string applied the same basis, sometimes with the keywords searched in the title, sometimes in the abstract. Thus, on these bases, the same work can be retrieved twice.
- **EC_3:** Lack of information about the work;
- **EC_4:** The work has already been selected by another source;
- **EC_5:** The work is not in English language.

Based on the inclusion and exclusion criteria, three stages were defined for the selection of works. The first was based on the analysis of keywords, title and abstract to decide whether the work may or may not be included. In the second stage, the introduction and conclusion were considered for analysis and third, the analysis was applied to the whole work.

For synthesis and extraction of data, some additional information to the research questions were collected, such as: which work focuses on web applications for generation of test data, whom authored the selected works, and what is the relation between the selected primary studies.

B. Conducting

After the protocol specification, we started to apply the search strings on the selected databases. Observe that this step requires, sometimes, and adaptation of the default search string to satisfy specific constraints of a particular database search engine. The complete set of search strings were omitted for sake of space, but they can be found in [20]. The application and adaptation of search strings happened from 8 to 11 October 2012, which returned all the control articles previously mentioned in Section III-A. In Table I, one can get the number of items returned for each search string in each database. Columns IC_n and EC_n correspond to the inclusion and exclusion criteria defined in Section III-A, respectively. Numbers in these columns represent the total of primary studies included or excluded from analysis based on that specific inclusion or exclusion criterion.

Identification and selection of the work was performed in three steps: reading the title, keywords and abstract; reading the introduction and conclusion, and reading the whole paper. In the first step, using the JabRef [21], each work was analyzed by two experts applying the all the inclusion and some exclusion criteria (EC_1, EC_2, and EC_3), defined in Section III-A, thus helping organizing and cataloging the works. This analysis took place in the order of application of search strings, first considering the ones applied on titles (ACM-1 and IEEE-1) and subsequently the ones applied on abstracts (ACM-2 and IEEE-2), resulting in 98 works for the inclusion criteria representing 16,39% of the total.

However redundant works between databases had not yet been identified, that is, the exclusion criterion (EC_4) had not yet been applied. After the application of EC_4 and reading the introduction and conclusion (Step 2) the number of selected papers was reduced to 59, corresponding to 9,87% of the total, as shown in the summary presented in Table I. Observe that there is no primary study written on language different than English, therefore, the exclusion criterion EC_5 was not applied.

In the final selection process, third stage, studies were analyzed completely and thereafter 39 primary studies were selected to compose the mapping, 6,52% of the 598 primary studies initially selected. It is observed that this reduction rate is consistent with other surveys in the area [6][22].

C. Analysis of the Results

Figure 1 presents the 39 selected primary studies organized by year as a directed graph. The arrows indicate a given primary study cite another one. Observe that from 1998 until the date of application of search strings, the majority of studies is concentrated in 2010, summing up 10 studies. However, the only primary study identified in 2001 was cited by 15 other works, and all studies from 2010 are referenced together by 8 other studies. Two studies that may be considered references

 ISBN: 978-1-61208-367-4

TABLE I. Results of the Second Analysis of the Primary Studies

Strings	IC1	IC2	IC3	IC4	EC1	EC2	EC3	EC4	Total IC (%)	Total EC (%)	Total
ACM-1	21	3	1	2	87	0	1	0	27 (23,48%)	88 (76,52%)	115
ACM-2	2	0	0	0	7	15	0	0	2 (8,34%)	22 (91,66%)	24
IEEE-1	10	1	0	1	53	0	2	13	12 (15,00%)	68 (85,00%)	80
IEEE-2	1	0	0	0	5	19	0	1	1 (3,84%)	25 (96,16%)	26
SCIENCE	4	1	0	0	151	0	25	1	5 (2,75%)	177 (97,25%)	182
SCOPUS	11	0	1	0	134	0	1	24	12 (7,02%)	159 (92,13%)	171
AMOUNT	49	5	2	3	437	34	29	39	59 (9,87%)	539 (90,13%)	598

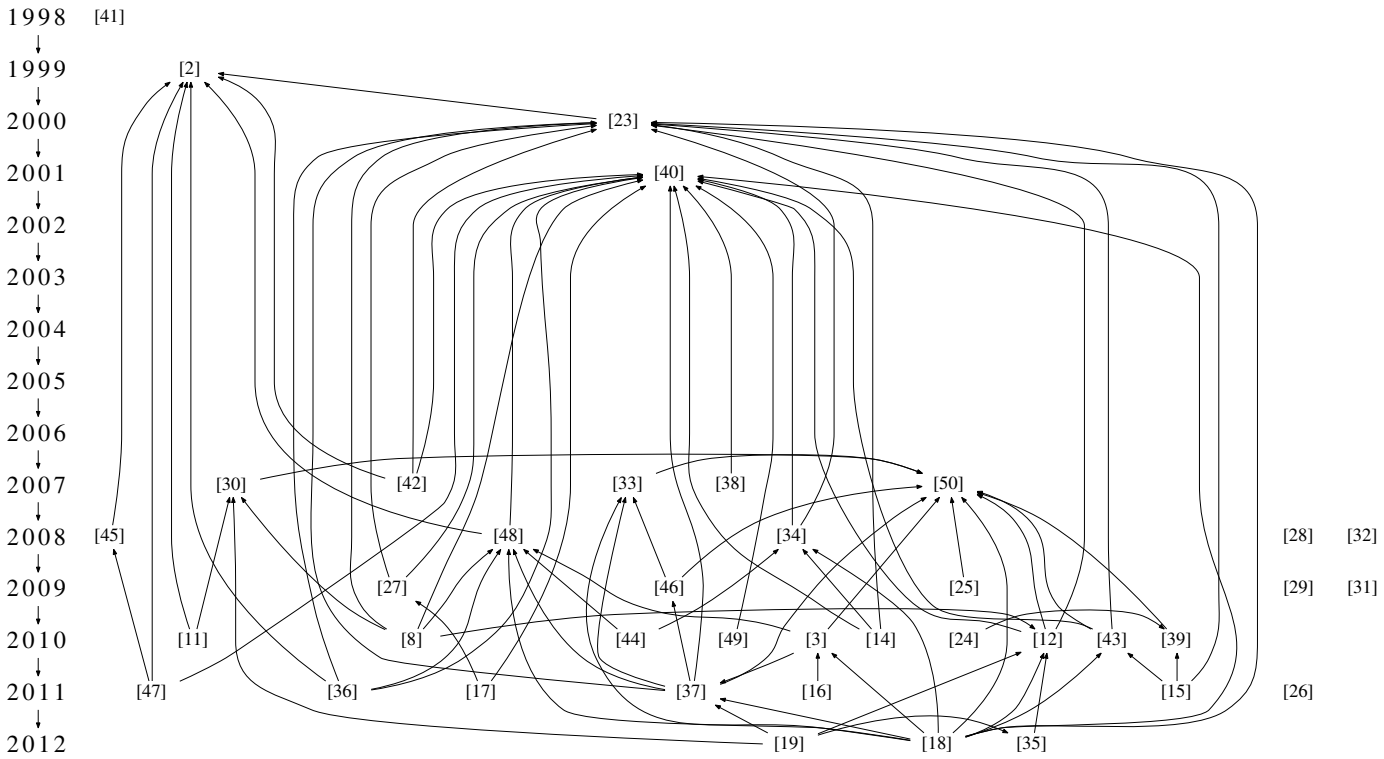


Figure 1. Distribution and Citation Between the Primary Studies

are the one authored by White and Almezen [23] and the one authored by Memon et. al. [2] with 11 and 15 citations each, respectively. The 39 selected primary studies involved 74 different authors of 27 affiliations (institutions) distributed in 13 countries. Most work within this context are authored by the same authors. An example is Dr. Atif M. Memon with participation, in approximately, 31% of the selected studies and can be considered a reference in the generation of test data from GUI. Highlight for the University of Maryland in the USA, appearing as an institution and country with more participation in the selected studies, 12 (30,8%) and 15 (38,5%), respectively.

Regarding the first research question RQ_1, the works do not identify a specific test criterion, but sometimes they mention which technique is used for generation of test data. Among the three traditional testing techniques, functional, structural, and fault-based, functional corresponds to 94,8%, since when the study did not mention what technique was used it was considered as functional since test data generation is based on GUI. Some works, however, besides the functional technique also use the source (structural technique) to guide its search technique or methodology for test data generation [15][26][27][48].

To answer the research question RQ_2, we analyzed the meta-heuristics and techniques generally used in studies for the generation of test data. In this case, the distribution of works was more uniform, with a small majority of 10 studies (25,6%) using Genetic Algorithms [3][14][26][32] [36][38][46][49].

Genetic algorithms are implemented as a computer simulation in which an initial population, in general randomly generated, representing a possible solution, is evolved to a better solution through iterations. The evolution occurs through generations. On each generation, the quality of the current solution in the population is evaluated, some individuals are selected for the next generation, and mutated or recombined to form a new population. The new population is then used as input to the next iteration of the algorithm.

Another meta-heuristic that was also used in three studies [16][18][34] was the Ant Colony Optimization. The algorithm was created by Marco Dorigo in 1992 [51] and was inspired in the behavior of ants searching for food. This is justified by the fact that a colony of insects are very organized and collective activities are carried out with self-organization. The idea is that ants move randomly in search of food, i.e., they conduct exploratory searches for possible solutions. Once one

 ISBN: 978-1-61208-367-4

finds food, it returns to the nest depositing pheromone. The greatest amount of pheromone means that more ants found this path and deposited the pheromone, increasing the likelihood of this being the best or the shortest way. Thus, this path became a solution that was optimized according to the level of pheromone found in the trail.

Other meta-heuristics for generating test data were also used, as is the case studies [14] and [35] which applied *Q-Lerning* and the work of Raul et. al. [44] that used Particle Swarm Optimization. In addition to meta-heuristics, other techniques have been identified as the work [17] and [27] who have used ontologies.

A fact that has been observed and that should be explored is that several studies [14][15][29] need an initial model of the GUI's application to perform the generation of test data (research question RQ_3). Just the work of Mariani et al. [19] employed automatic generation of a model and produced the test data incrementally by traversing the GUI model of the application under test, requiring no human intervention. They used the *Q-Lerning*, a tool from the AI area that learns to interact with the application under test and to stimulate their functionality.

Responding to the fourth research question RQ_4, some tools that assist in generating test data were identified during the mapping. Most tools are complementary, i.e., they allow to obtain better results when combined. One of the most used tool in the selected studies was *GUI Testing Framework* (GUITAR) [45] which was used in 42% of the selected studies. This is a project supported by *National Science Foundation*, aiming at simplifying GUI testing by automatically generating test data to test the functionality of the program under test via the GUI. This tool is divided into four components that represent its main functions: *GUIRipper* that extracts information from the GUI of the application under testing; the *GUIStructure2Graph* that builds an event flow graph (EFG) with the GUI elements; the *TestCaseGenerator* which generates a set of test data based on the EFG (but without the use of meta-heuristics); and *GUIReplayer* responsible for running the program with the generated suite of tests. One of the studies that applied the tool was performed by Huang et al. [3]. They used genetic algorithms to correct invalid test data sets. The work consisted of two steps: generate a set of test and repair the test set containing viable sequences for this, used the EFG model generated by GUITAR.

The work of Mariani et al. [19] aims to implement and to evaluate a technique for generating test data focusing on interactive applications, i.e., applications that interact with users through a GUI or Web. The technique and tool developed and used are called AutoBlackTest, that works with the generation of a model and produces the test data incrementally by exercising the application under test. For this, it uses Q-Lerning, an optimization techniques in the area of Artificial Intelligence, that learns how to interact with the application under test to stimulate their functionality.

An important feature of this work is that the *AutoBlackTest* does not depend on an initial set for execution. The vast majority of current techniques depends on this data set and to generate GUI testing its works in two phases [37][40]: generates a model of the sequences of events that can be produced through the interaction with the GUI application under test; and generates a set of test data that covers sequences in the model.

The effectiveness of these techniques depends on the integrity of the original model. When the initial model is obtained by stimulation of the application under test with a simple sampling strategy that uses a subset of GUI actions to navigate and analyze the windows, the derived model is partial and incomplete [19]. Thus, the test data generated can ignore many interactions and windows not discovered in the initial phase.

To evaluate the proposal, Mariani et. al. [19] carried out a comparative empirical evaluation between *AutoBlackTest* with the GUITAR tool using four applications for *desktop* computers. In the empirical comparison between *AutoBlackTest* and GUITAR, when applied in sessions with 12 hours of testing, was conclusive that *AutoBlackTest* can generate test data that reach a higher code coverage and also reveals more flaws than GUITAR.

Finally, answering the research question RQ_5, most of the selected studies, approximately 95% use desktop application for generating test data from the GUI. Only the studies [26][31][32] applies the proposal in a Web context, thus showing that much can still be done in this area.

IV. Conclusion

This study applied a systematic mapping of the literature between the years 1998 to 2012, on application of techniques for generating test data from the GUI of the application under test. 39 primary studies from different sources between regular and high-level conferences were selected, corresponding to 6,52% of the total number of studies identified by the search application. It was found that this percentage is justified due to two reasons: (1) there are many works that apply, evaluate and propose techniques to generate test data, but not using as reference the GUI; and (2) some works focus on generating test data to test the GUI itself and not use it as input for the generation of test data.

With respect to the five research question we investigated, we found that, in general, the proposed testing generation techniques employed most functional testing criteria for test set quality evaluation (RQ_1). In terms of meta-heuristic (RQ_2), Genetic Algorithm is employed in 10 out of 39 the primary studies, followed by ant colony employed in 3 out of 39 studies, and q-learn which were employed 2 out of 39 primary studies (RQ_3). In terms of automation (RQ_4) GUITAR (a tool to test GUI desktop applications) was used in 42% of the primary studies, reinforcing the result obtained by RQ_5 that almost all studies were performed in the context of desktop applications.

Therefore, based on primary studies identified and answers to the research questions, we can highlight some research areas in the context of testing from GUI to be explored, which is the main purpose of a systematic mapping study:

- development of a software environment that allows to abstract the GUI model automatically, providing subsidies so that test data can be run at any time and if there is a change or modification in the GUI, the model can be updated and reevaluated at any time;

 ISBN: 978-1-61208-367-4

- conduction of experimental studies to compare the different test data generation techniques, identifying the main characteristics of each one;
- definition of a strategy to reduce the cost and increase efficiency in the generation of test data from GUI;
- adaptation of the representation of the techniques presented for generating test data from GUIs for Web applications; and
- conduction of systematic reviews considering more specific research questions about the use of meta-heuristics to support the automation of test data generation.

ACKNOWLEDGMENT

The authors would like to thank the Brazilian funding agencies Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Fundação de Amparo à Pesquisa do Estado de Goiás (FAPEG), and Fundação de Apoio ao Desenvolvimento do Ensino, Ciência e Tecnologia do Estado de Mato Grosso do Sul (FUNDECT) which support this work.

REFERENCES

[1] G. J. Myers and C. Sandler, The Art of Software Testing. John Wiley & Sons, 2004.

[2] A. Memon, M. Pollack, and M. Soffa, "Using a goal-driven approach to generate test cases for guis," in Software Engineering, 1999. Proceedings of the 1999 International Conference on, May 1999, pp. 257–266.

[3] S. Huang, M. B. Cohen, and A. M. Memon, "Repairing gui test suites using a genetic algorithm," in Proceedings of the 2010 Third International Conference on Software Testing, Verification and Validation, ser. ICST'10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 245–254. [Online]. Available: http://dx.doi.org/10.1109/ICST.2010.39

[4] Y. Zhang, "Sbse repository," Página Web, Aug. 2013, disponível em: http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/. Acesso em: 12/12/2013.

[5] B. Kitchenham, S. Charters, D. Budgen, P. Brereton, M. Turner, S. Linkman, M. Jorgensen, E. Mendes, and G. Visaggio, "Guidelines for performing systematic literature reviews in software engineering," Software Engineering Group – School of Computer Science and Mathematics – Keele University, Keele, Staffs, ST5 5BG, UK, Tech. Rep. EBSE-2007-01, Jul. 2007.

[6] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in Proceedings of the 12th international conference on Evaluation and Assessment in Software Engineering, ser. EASE'08. Swinton, UK, UK: British Computer Society, 2008, pp. 68–77. [Online]. Available: http://dl.acm.org/citation.cfm?id=2227115.2227123

[7] J. Biolchini, P. G. Mian, A. C. C. Natali, and G. H. Travassos, "Systematic review in software engineering," Systems Engineering and Computer Science Dept. - COPPE/UFRJ, Rio de Janeiro/RJ - Brazil, Technical Report RT-ES 679/05, 2005.

[8] L. Chen, M. A. Babar, and H. Zhang, "Towards an evidence-based understanding of electronic data sources," in International Conference on Evaluation and Assessment in Software Engineering (EASE2010). Keele, UK: BCS, Apr. 2010.

[9] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, "A systematic review of the application and empirical investigation of search-based test case generation," IEEE Trans. Softw. Eng., vol. 36, no. 6, Nov. 2010, pp. 742–762. [Online]. Available: http://dx.doi.org/10.1109/TSE.2009.52

[10] A. M. Memon, M. E. Pollack, and M. L. Soffa, "Hierarchical gui test case generation using automated planning," IEEE Trans. Soft. Eng., vol. 27, no. 2, Feb. 2001, pp. 144–155. [Online]. Available: http://dx.doi.org/10.1109/32.908959

[11] M. Cunha, A. Paiva, H. Ferreira, and R. Abreu, "Pettool: A pattern-based gui testing tool," in Software Technology and Engineering (ICSTE), 2010 2nd International Conference on, vol. 1, Oct. 2010, pp. 202–206.

[12] X. Yuan and A. M. Memon, "Generating event sequence-based test cases using gui runtime state feedback," IEEE Trans. Softw. Eng., vol. 36, no. 1, Jan. 2010, pp. 81–95. [Online]. Available: http://dx.doi.org/10.1109/TSE.2009.68

[13] ——, "Iterative execution-feedback model-directed gui testing," Inf. Softw. Technol., vol. 52, no. 5, May 2010, pp. 559–575. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2009.11.009

[14] A. Rauf, S. Anwar, M. A. Jaffer, and A. A. Shahid, "Automated gui test coverage analysis using ga," in Proceedings of the 2010 Seventh International Conference on Information Technology: New Generations, ser. ITNG'10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1057–1062. [Online]. Available: http://dx.doi.org/10.1109/ITNG.2010.95

[15] S. Arlt, C. Bertolini, and M. Schäf, "Behind the scenes: An approach to incorporate context in gui test case generation," in Proceedings of the 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, ser. ICSTW'11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 222–231. [Online]. Available: http://dx.doi.org/10.1109/ICSTW.2011.70

[16] S. Bauersfeld, S. Wappler, and J. Wegener, "An approach to automatic input sequence generation for gui testing using ant colony optimization," 2011, pp. 251–252.

[17] H. Li, H. Guo, F. Chen, H. Yang, and Y. Yang, "Using ontology to generate test cases for gui testing," Int. J. Comput. Appl. Technol., vol. 42, no. 2/3, Feb. 2011, pp. 213–224. [Online]. Available: http://dx.doi.org/10.1504/IJCAT.2011.045407

[18] Y. Huang and L. Lu, "Apply ant colony to event-flow model for graphical user interface test case generation," IET Software, vol. 6, no. 1, 2012, pp. 50–60.

[19] L. Mariani, M. Pezze, O. Riganelli, and M. Santoro, "Autoblacktest: Automatic black-box testing of interactive applications," Software Testing, Verification, and Validation, 2008 International Conference on, vol. 0, 2012, pp. 81–90.

[20] R. F. Jorge, "Geração de dados de teste a partir de gui: Um mapeamento sistemático," 2013, 24 junho 2014. [Online]. Available: http://www.inf.ufg.br/~auri/icsea2014/

[21] JabRef, "Ferramenta JabRef," Página do Projeto, Oct. 2013, disponível em: http://jabref.sourceforge.net/. Acesso em: 12/12/2013.

[22] E. EngstrÃm and P. Runeson, "Software product line testing a systematic mapping study," Information and Software Technology, vol. 53, 2011, pp. 2–13.

[23] L. White and H. Almezen, "Generating test cases for gui responsibilities using complete interaction sequences," in Proceedings of the 11th International Symposium on Software Reliability Engineering, ser. ISSRE'00. Washington, DC, USA: IEEE Computer Society, 2000, pp. 110–124. [Online]. Available: http://dl.acm.org/citation.cfm?id=851024.856239

[24] C. Bertolini and A. Mota, "A framework for gui testing based on use case design," in Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops, ser. ICSTW'10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 252–259. [Online]. Available: http://dx.doi.org/10.1109/ICSTW.2010.37

[25] S. Qian and F. Jiang, "An event interaction structure for gui test case generation," in Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on, Aug. 2009, pp. 619–622.

[26] X. Peng and L. Lu, "A new approach for session-based test case generation by ga," in Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on, May 2011, pp. 91–96.

[27] H. Li, F. Chen, H. Yang, H. Guo, W. C.-C. Chu, and Y. Yang, "An ontology-based approach for gui testing," in Proceedings of the 2009 33rd Annual IEEE International Computer Software and Applications Conference - Volume 01, ser. COMPSAC'09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 632–633. [Online]. Available: http://dx.doi.org/10.1109/COMPSAC.2009.92

 ISBN: 978-1-61208-367-4

[28] X. Zhu, B. Zhou, J. Li, and Q. Gao, "A test automation solution on gui functional test," 2008, pp. 1413–1418. [Online]. Available: http://goo.gl/cxZ623

[29] Y. Hou, R. Chen, and Z. Du, "Automated gui testing for j2me software based on fsm," in Proceedings of the 2009 International Conference on Scalable Computing and Communications; Eighth International Conference on Embedded Computing, ser. SCALCOM-EMBEDDEDCOM'09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 341–346. [Online]. Available: http://dx.doi.org/10.1109/EmbeddedCom-ScalCom.2009.67

[30] P. A. Brooks and A. M. Memon, "Automated gui testing guided by usage profiles," in Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, ser. ASE'07. New York, NY, USA: ACM, 2007, pp. 333–342. [Online]. Available: http://doi.acm.org/10.1145/1321631.1321681

[31] A. Shahzad, S. Raza, M. Azam, K. Bilal, Inam-Ul-haq, and S. Shamail, "Automated optimum test case generation using web navigation graphs," 2009, pp. 427–432. [Online]. Available: http://goo.gl/tiHCG8

[32] S. H. Kuk and H. S. Kim, "Automatic generation of testing environments for web applications," in Proceedings of the 2008 International Conference on Computer Science and Software Engineering - Volume 02, ser. CSSE'08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 694–697. [Online]. Available: http://dx.doi.org/10.1109/CSSE.2008.1026

[33] X. Yuan, M. Cohen, and A. M. Memon, "Covering array sampling of input event sequences for automated gui testing," in Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, ser. ASE'07. New York, NY, USA: ACM, 2007, pp. 405–408. [Online]. Available: http://doi.acm.org/10.1145/1321631.1321695

[34] Y. Lu, D. Yan, S. Nie, and C. Wang, "Development of an improved gui automation test system based on event-flow graph," in Proceedings of the 2008 International Conference on Computer Science and Software Engineering - Volume 02, ser. CSSE'08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 712–715. [Online]. Available: http://dx.doi.org/10.1109/CSSE.2008.1336

[35] G. Becce, L. Mariani, O. Riganelli, and M. Santoro, "Extracting widget descriptions from guis," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 7212 LNCS, 2012, pp. 347–361. [Online]. Available: http://goo.gl/jmLntA

[36] A. Rauf, A. Jaffar, and A. Shahid, "Fully automated gui testing and coverage analysis using genetic algorithms," International Journal of Innovative Computing, Information and Control, vol. 7, no. 6, 2011, pp. 3281–3294. [Online]. Available: http://goo.gl/zsRadh

[37] X. Yuan, M. B. Cohen, and A. M. Memon, "Gui interaction testing: Incorporating event context," IEEE Trans. Softw. Eng., vol. 37, no. 4, Jul. 2011, pp. 559–574. [Online]. Available: http://dx.doi.org/10.1109/TSE.2010.50

[38] I. Alsmadi and K. Magel, "Gui path oriented test generation algorithms," in Proceedings of the Second IASTED International Conference on Human Computer Interaction, ser. IASTED-HCI'07. Anaheim, CA, USA: ACTA Press, 2007, pp. 216–219. [Online]. Available: http://dl.acm.org/citation.cfm?id=1698252.1698291

[39] C. Bertolini, A. Mota, E. Aranha, and C. Ferraz, "Gui testing techniques evaluation by designed experiments," in Proceedings of the 2010 Third International Conference on Software Testing, Verification and Validation, ser. ICST'10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 235–244. [Online]. Available: http://dx.doi.org/10.1109/ICST.2010.41

[40] A. M. Memon, M. L. Soffa, and M. E. Pollack, "Coverage criteria for gui testing," in Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering, ser. ESEC/FSE-9. New York, NY, USA: ACM, 2001, pp. 256–267. [Online]. Available: http://doi.acm.org/10.1145/503209.503244

[41] A. Beer, S. Mohacsi, and C. Stary, "Idatg: an open tool for automated testing of interactive software," in Computer Software and Applications Conference, 1998. COMPSAC'98. Proceedings. The Twenty-Second Annual International, Aug. 1998, pp. 470–475.

[42] M. Hayat and N. Qadeer, "Intra component gui test case generation technique," in Information and Emerging Technologies, 2007. ICIET 2007. International Conference on, Jul. 2007, pp. 1–5.

[43] X. Yuan and A. M. Memon, "Iterative execution-feedback model-directed gui testing," Inf. Softw. Technol., vol. 52, no. 5, May 2010, pp. 559–575. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2009.11.009

[44] R. Abdul, N. Ejaz, Q. Abbas, S. Rehman, and A. Shahid, "Pso based test coverage analysis for event driven software," 2010, pp. 219–224. [Online]. Available: http://goo.gl/7mdpDZ

[45] D. R. Hackner and A. M. Memon, "Test case generator for guitar," in Companion of the 30th international conference on Software engineering, ser. ICSE Companion'08. New York, NY, USA: ACM, 2008, pp. 959–960. [Online]. Available: http://doi.acm.org/10.1145/1370175.1370207

[46] X. Yuan, M. Cohen, and A. Memon, "Towards dynamic adaptive automated test generation for graphical user interfaces," in Software Testing, Verification and Validation Workshops, 2009. ICSTW '09. International Conference on, Apr. 2009, pp. 263–266.

[47] K. C. Chuang, C. S. Shih, and S. H. Hung, "User behavior augmented software testing for user-centered gui," in Proceedings of the 2011 ACM Symposium on Research in Applied Computation, ser. RACS'11. New York, NY, USA: ACM, 2011, pp. 200–208. [Online]. Available: http://doi.acm.org/10.1145/2103380.2103421

[48] Q. Xie and A. Memon, "Using a pilot study to derive a gui model for automated testing," ACM Transactions on Software Engineering and Methodology, vol. 18, no. 2, 2008, pp. 1–35. [Online]. Available: http://goo.gl/cXsdQi

[49] I. Alsmadi, "Using genetic algorithms for test case generation and selection optimization," 2010, pp. 1–4. [Online]. Available: http://goo.gl/xNZhRY

[50] X. Yuan and A. M. Memon, "Using gui run-time state as feedback to generate test cases," in Proceedings of the 29th international conference on Software Engineering, ser. ICSE'07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 396–405. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2007.94

[51] M. Dorigo and L. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," Evolutionary Computation, IEEE Transactions on, vol. 1, no. 1, 1997, pp. 53–66.

 ISBN: 978-1-61208-367-4

Mapping of State Machines to Code: Potentials and Challenges

Mehrdad Saadatmand
Mälardalen Real-Time Research Centre (MRTC),
Mälardalen University,
Västerås, Sweden
mehrdad.saadatmand@mdh.se
& Alten AB,
Sweden
mehrdad.saadatmand@alten.se

Antonio Cicchetti
Mälardalen Real-Time Research Centre (MRTC),
Mälardalen University,
Västerås, Sweden
antonio.cicchetti@mdh.se

***Abstract*—There is a big number of testing methods which are based on the state machine formalism. State machines serve as a strong means in behavior modeling of computer systems. However, this strength is lost as we go down the abstraction level from models to code. This is essentially due to the inherent semantic gap between state machine models and code, and that it is generally a challenging task to track states and transitions at runtime. In this paper, we discuss the benefits and challenges of having a mechanism for mapping state machines to code. The main intention with such a mechanism is to enable tracking of state changes at runtime. As we explain in this paper, the mapping provides for several important testing features such as verifying the actual runtime behavior of the system against its state machine models. Its importance becomes further emphasized remembering that model-based static analysis techniques rely on models as the source of information and design assumptions, and therefore, any mismatch between the actual behavior of the system and its models can also imply invalidation of the analysis results.**

Keywords–State Machines; Runtime Verification; Behavioral Modeling.

I. INTRODUCTION

Applying Model-Based Development (MBD) methodology [1][2] helps to cope with the ever-increasing complexity of computer systems. It does so by raising the abstraction level, enabling analysis at earlier phases of development and automatic code generation. In Model-Based Testing (MBT), models serve as an explicit representation of the intended behavior of a system from which test cases are generated [3] [4]. Test cases are then executed to detect failures and to verify if the intended and actual behavior of a system differ.

State machines [5] serve as a modeling formalism for behavioral description of different types of systems (particularly reactive systems) and are used extensively in model-based testing techniques. State machine models can thus capture the expected behavior of a system. In testing the behavioral aspects, it is necessary to be able to determine whether the runtime behavior of the system is in compliance with its specified behavior represented and described using state machine models. This is regardless if the code is manually developed or automatically generated from system models. For this purpose, there needs to be a mechanism to establish a mapping between state machines and code in order to verify that the code at runtime is actually behaving correctly according to the state machine model in terms of its internal states and transitions.

In [6], we have introduced and developed an approach for testing the behavior of automotive embedded systems, by comparing against the Timed Automata (TA) [7][8] specification models that are used to describe the internal behavior of system components, modeled using EAST-ADL language [9] (the term *state machine* is used in this paper as a synonym to also refer to a timed automaton whenever the main concern is only the states and transitions in the model regardless of the timing specifications). In the approach, timed automata models are analyzed to identify if different properties hold or not. As a result, a trace is generated consisting of a sequence of states and transitions serving as a witness or counter-example of the performed analysis. To verify the actual behavior of the system, it is then checked at runtime if the system traverses and goes through the exact order of states and transitions as in the trace file. To achieve this, as part of the approach we have defined a mechanism for mapping state machines to code. The mapping is currently done manually, which is not that scalable especially when the size of code grows. In this paper, we focus on the mapping mechanism and discuss its importance and the capabilities it provides for testing the behavior of systems based on our findings in [6]. We introduce our specific way of implementing the mapping mechanism along with its features and limitations. Moreover, other different possible solutions to implement such a mapping mechanism and the related challenges are also described and identified. In short, the main intention with this paper is to highlight the benefits of having a mapping mechanism between state machine models and code (as part of our research project results); particularly that establishing such a mapping can require early design decisions and following certain rules in the code to enable tracking states and transitions at runtime.

The remainder of the paper are structured as follows. In Section II, background context and motivation of this work is presented. Related work and possible solutions for the mapping mechanism, along with the challenges and potentials of having such a mapping mechanism are discussed in detail in Section III. Finally, Section IV concludes the paper and there, we also discuss the future directions of this work.

II. BACKGROUND & MOTIVATION

This work has been performed in the scope of the the Combined Model-based Analysis and Testing of Embedded Systems (MBAT) European project [10] consisting of 38 project partners. One of the main goals in MBAT is to provide

 ISBN: 978-1-61208-367-4

a more efficient and effective Verification & Validation solution for embedded systems by exploiting the synergy between model-based analysis and testing. Brake-By-Wire (BBW) system from Volvo is one of the industrial use-cases that are addressed in MBAT. In a BBW system, mechanical parts and hydraulic connections between the brake pedal and each wheel brake are replaced by electronic sensors and actuators. Anti-lock Braking System (ABS) is usually an inherent functionality provided by BBW systems [11] whose purpose is to prevent the locking of wheels by controlling braking based on *slip rate*. There is a threshold for the slip rate beyond which the brake actuator is released and no brake is applied (otherwise the requested brake torque is used).

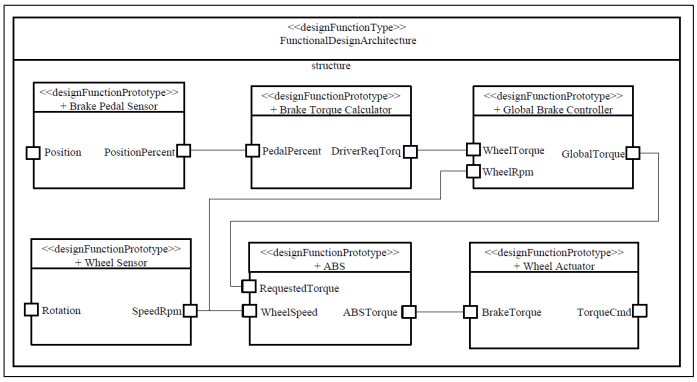


Figure 1. Components composing a BBW system [6].

The system is modeled using EAST-ADL. In Figure 1, a simplified model of the system for only one wheel is depicted. The internal behavior of system components are captured and represented using Timed Automata (TA). Since BBW is a real-time system with different timing requirements, the use of TA models enable to also include timing specifications. Considering the purpose of a BBW system, it is generally considered as a safety-critical, distributed real-time (embedded) systems. A timed automata model, designed in UPPAAL tool [12], describing the internal behavior of the ABS component of BBW system is shown in Figure 2. In this model, y is a clock whose specification on the states indicates the amount of time units that can be spent in each state (non-deterministically, between 0 and the specified value) before a transition has to be made to the next state. These timing specifications are naturally derived from high level timing requirements of the BBW system and its components. The values in the TA model here are just samples, and the exact values for each implementation of the BBW system might be different.

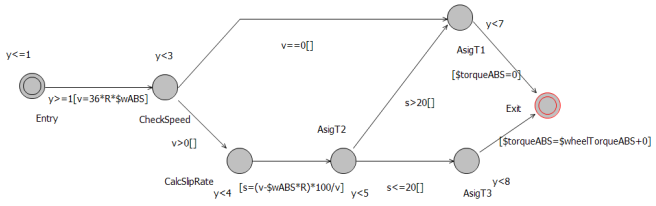


Figure 2. Timed automata model of the ABS component [6].

A. Testing Goals

We have developed a testing methodology [6] in order to verify the runtime behavior of the BBW system against the desired behavior represented in the form of state machine and automata models. To do so, test cases are generated from automata models as UPPAAL trace files. These traces serving as *abstract* test cases are then transformed into *concrete* ones, which are essentially executable test scripts. By executing the concrete test cases the runtime behavior of the system is verified in terms of state changes. In other words, it is checked that the order of states of the system at runtime matches what is specified in the models.

Some of the steps that constitute the approach are as follows:

- Based on the automata models, C/C++ enumerations (enum) that represent each state machine and their internal states are generated. These enumeration structures are stored in a C/C++ file along with the definition of a helper function called `set_state(StateMachine,State)`. The file is then included in the implementation code of the target application (i.e., to be tested).
- The states in the automata model are mapped to the code using the above helper function. This is done by adding calls to the `set_state()` helper function at places in the code where a state change occurs. The helper function basically logs the new state belonging to the specified state machine and thus enables to keep track of state changes at runtime.
- According to the automata model, a test script is generated which verifies that the order of state changes (logged using the helper function) match the model. If so, then the result of the test is determined as *pass*, otherwise a *fail* verdict is decided.

This helps to gain more confidence that the behavior of the system is actually as specified and expected at the modeling level. One of the motivations behind our approach is that the models are used for different types of model-based analysis. If the runtime behavior of the system deviates from and does not match the behavioral models, the result of the analyses that have been performed assuming such behavioral models will be violated and not valid anymore.

III. STATE MACHINE MAPPING

A. Challenges

To provide a mechanism for tracking state changes at runtime, the `set_state(StateMachine,State)` helper function that was introduced in the previous section is used to *map* state machine models to code. This mapping step is needed to keep track of different states and how they change at runtime, which is currently done in a manual way. Figure 3 shows how this mapping is done by *annotating* the code and adding calls to the helper function in it. The code shown here is C/C++ code for the ABS component written on OSE Real-Time Operating System (RTOS), which is a commercial and industrial real-time operating system developed by Enea [13]. OSE offers the concept of direct and asynchronous message passing for communication and synchronization between tasks using *send* and *receive* APIs.

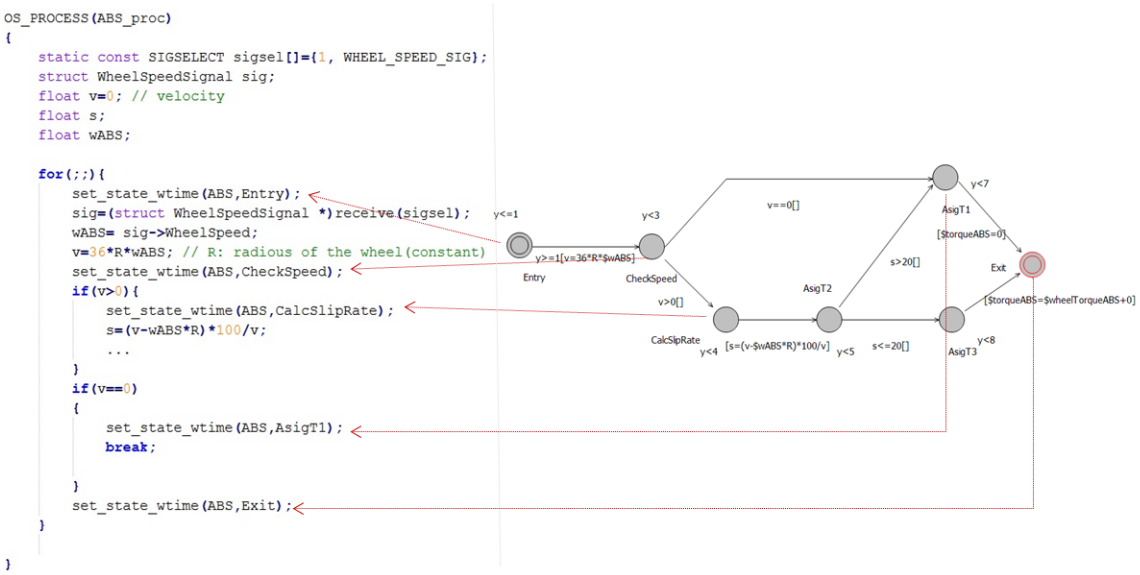


Figure 3. Mapping of states to the code (ABS function) [6].

As mentioned, the mapping step is currently done manually. While this works fine for systems with a small code base, it has a big impact for the scalability of the general approach. From this perspective, the manual mapping step can be considered a bottleneck in the automation of the whole approach. Therefore, there is a need and big interest in automating the mapping step by finding a solution to map state machine models to code in order to track state changes at runtime.

B. Potentials

So far, it was discussed how having such a mapping mechanism can help to track the runtime behavior of the system with respect to its internal state changes. Information on internal state changes basically provides an insight about execution flow in the system. On the other hand, state machines used in the modeling phase represent the desired and expected behavior of the system. The mapping mechanism enables to compare the actual versus expected behavior and identify violations. Moreover, it also becomes possible to identify at which state and during which transition a violation has occurred. This, in turn, can help with debugging and better pinpointing the root cause of the problem than the case where there is no such mapping; hence identifying the *vicinity* and localization of potential defects. We have discussed and demonstrated these features in detail in [14], as a method for checking architectural consistency.

Moreover, having the mapping mechanism and being able to track state changes at runtime brings along other interesting testing capabilities. One of such capabilities is to verify timing properties and clock constraints in real-time systems. We have introduced and demonstrated it in [15]. The idea is basically that not only each state change is recorded, but they are also time stamped. This way, it becomes possible to measure the time difference between each pair of states during the actual execution of the system and at runtime. This information is then used to compare the timing behavior of the system versus the timing and clock constraints that are specified in the timed automata models. For example, a clock constraint can be defined in the timed automata model specifying that the system may spend time and remain in a state only for a certain period of time and then it has to make the transition to the next state. Such timing requirements are of great importance in designing real-time systems, but are also hard to actually test at such granularity and level of detail. Timed automata are a formal way of capturing such timing requirements and constraints, which are then used also for analyzing the temporal correctness of the system design. However, despite performing static analysis, at runtime situations may still occur, which lead to the violation of assumptions that have been taken into account for performing the analyses; hence invalidation of the analysis results [16][17]. This further emphasizes the need to be able to actually test a system with respect to its extra-functional properties; and particularly in this case, timing properties which are not as easy and straightforward to perform as functional testing. The approach we have introduced in [15] is one solution towards this purpose, which is based on the state machine mapping mechanism in order to test clock constraints in real-time systems.

C. Related Work & Possible Solutions

There is not much discussion in the literature on establishing a mapping and relationship between state machines and code for the purposes mentioned in this paper. Walkinshaw et al. in [18] discuss the problem of rarely maintaining state machine models during software development by emphasizing and drawing attention to their importance and role in state-based testing techniques. They introduce an approach based on symbolic execution to reverse engineer state transitions from code. However, what we discussed here can be considered as opposite of their approach and with the purpose of tracking state changes at runtime; which their introduced reverse engineering approach does not provide. Moreover, the mapping from the direction of state machine models to code and then runtime tracking of state changes helps to identify situations where the behavior of code deviates from what is

 ISBN: 978-1-61208-367-4

specified by the model as the correct behavior. In contrast, merely extracting state machine models from the code just creates the model of how the code behaves, which can contain and represent a wrong behavior. What might be possible here is to reverse engineer state machine model of the code and then compare it with a designed state machine model representing the correct and expected behavior that the system should conform to. In [19], the DiscoTect method and tool are introduced. DiscoTect engines observe and filter system events during execution and at runtime to discover and construct the architecture of the system based on derived states and transitions. The ultimate goal in DiscoTect is to highlight inconsistencies between the implementation and intended architecture by providing the discovered architecture, which can then be compared with intended one. SMArTIC [20] is also an architecture and method for specification mining which is similar to DiscoTect, particularly that the discovered system specification is derived in the form of a finite state automaton. Other examples of such dynamic analysis techniques that derive state machine models from actual program executions are ADABU [21] and GK-tail [22]. The latter aims to capture the interplay between data values and component interactions by annotating state machine models with conditions on data values. FSMGen tool which is introduced in [23] utilizes a symbolic execution technique to statically analyze TinyOS program codes in order to derive state machine models of the system. The advantage that static analysis methods have over dynamic ones (e.g., the ones mentioned above) is that dynamic approaches can capture and analyze only particular runs of an application, while applications can generally have an infinite number of execution traces. All these mentioned approaches try to construct state machine models from code. However, as described earlier, in this paper, our focus is on the other direction which is from existing models (used in model-based analysis) to code and establishing mapping between them.

Another approach to enable tracking of state changes at runtime could be to implement or generate the code in the form of a state machine. In other words, the code is originally designed and written in the form of states and transitions; i.e., an implementation and code representation of the state machine. The Windows Workflow Foundation [24] provides tracking APIs, which make it (easily) possible to implement such an approach as demonstrated in [25]. Another example of this approach could be to have a variable to keep the current state and a switch-case structure (in C/C++) to choose execution blocks based on its value. In [26], where we have presented a more complete and extended version of our testing methodology, this approach is used. In other words, the code contains necessary variables to keep track of different states at runtime. A feature of this mapping approach is that if the code is automatically generated from the models, it can be made to insert and include the necessary variables to keep track of states and transitions as part of the code generation process.

Finally, automation of the manual mapping approach that was introduced in previous sections of the paper can serve as another solution. This requires an 'intelligent' tool, which goes through the code and tries to identify parts that match a state from the model (e.g., based on the guards, actions, and other information in the state machine model). The accuracy of such a tool needs to be considered carefully. For this reason, it may be made as a semi-automatic tool with user interactions to confirm whenever, for example, several matching points are detected for a state (false positives). A feature of this approach though is that it can be very helpful when there is already some code available (as opposed to the code generation approach discussed above), for instance, in legacy systems.

An advantage of the mapping approach in general is that the instrumentation of the code that is done to achieve the mapping can be done just to test the system and removed afterwards from the final product and before the actual deployment. The impacts of such instrumentation and how it may affect test results, particularly, in real-time systems need to also be taken into account, as we have discussed with more details in [15]. Moreover, it should be investigated if a state change always corresponds to only one location in the code, particularly when the target system is parallel or distributed, e.g., in multicore scenarios. In other words, the mapping in some systems might not always be one-to-one but also one-to-many.

IV. Conclusion

In this paper, we discussed the idea of mapping state machine models to code to enable tracking state changes at runtime. Moreover, the advantages and potentials that such a mapping can offer for testing were also presented along with the possible implementation solutions as well as the challenges that exist in implementing it. A manual establishment of mapping between state machines and code is currently being considered as part of a testing methodology for the Volvo's Brake-By-Wire use-case in the MBAT European project. However, the main challenge is that while such a manual mapping might work for a small system, it will not be scalable for systems with large code bases, and therefore, needs to be automated. In summary, the main goal of this paper has been to highlight the benefits and uses of having the mapping mechanism, discuss its feasibility, and encourage research on methods for automatic establishment of the mapping as well as its further use in testing.

V. Acknowledgements

This work has been supported by the MBAT European Project [10] and also through the ITS-EASY industrial research school [27]. The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement no 269335 (see Article II.9. of the JU Grant Agreement) and from the Swedish Governmental Agency for Innovation Systems (VINNOVA). We would also like to thank Raluca Marinescu and Dr. Cristina Seceleanu for their technical tips and support for this work.

References

[1] B. Selic, "The pragmatics of model-driven development," Software, IEEE, vol. 20, no. 5, Sept 2003, pp. 19–25.

[2] J. Bezivin, "On the unification power of models," Software Systems Modeling, vol. 4, no. 2, 2005, pp. 171–188.

[3] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," Software Testing, Verification and Reliability journal, vol. 22, no. 5, Aug. 2012, pp. 297–312.

[4] A. Pretschner, W. Prenninger, S. Wagner, C. Kühnel, M. Baumgartner, B. Sostawa, R. Zölch, and T. Stauner, "One evaluation of model-based testing and its automation," in Proceedings of the 27th international conference on Software engineering, ser. ICSE, New York, USA, 2005, pp. 392–401.

[5] T. S. Chow, "Testing software design modeled by finite-state machines," IEEE Trans. Softw. Eng., vol. 4, no. 3, May 1978, pp. 178–187.

 ISBN: 978-1-61208-367-4

[6] R. Marinescu, M. Saadatmand, A. Bucaioni, C. Seceleanu, and P. Pettersson, "EAST-ADL Tailored Testing: From System Models to Executable Test Cases," Mälardalen University, Technical Report ISSN 1404-3041 ISRN MDH-MRTC-278/2013-1-SE, August 2013.

[7] R. Alur and D. L. Dill, "A theory of timed automata," Theoretical Computer Science, vol. 126, no. 2, 1994, pp. 183 – 235.

[8] J. Bengtsson and W. Yi, "Timed automata: Semantics, algorithms and tools," in In Lecture Notes on Concurrency and Petri Nets, ser. Lecture Notes in Computer Science vol 3098, W. Reisig and G. Rozenberg, Eds. Springer–Verlag, 2004.

[9] The ATESST Consortium, "EAST-ADL Profile Specification." www.atesst.org, Accessed: August 2014.

[10] MBAT Project: Combined Model-based Analysis and Testing of Embedded Systems, http://www.mbat-artemis.eu/home/, Accessed: August 2014.

[11] S. Anwar, "An anti-lock braking control system for a hybrid electromagnetic/electrohydraulic brake-by-wire system," in American Control Conference, 2004. Proceedings of the 2004, vol. 3, 2004, pp. 2699–2704.

[12] G. Behrmann, R. David, and K. G. Larsen, "A tutorial on Uppaal 4.0," http://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf, November 2006.

[13] Enea, http://www.enea.com, Accessed: August 2014.

[14] M. Saadatmand, D. Scholle, C. W. Leung, S. Ullström, and J. F. Larsson, "Runtime verification of state machines and defect localization applying model-based testing," in Workshop on Software Architecture Erosion and Architectural Consistency (SAEroCon) - Proceedings of the WICSA 2014 Companion Volume, ser. WICSA '14 Companion. ACM, 2014, pp. 6:1–6:8.

[15] M. Saadatmand and M. Sjodin, "Testing of timing properties in real-time systems: Verifying clock constraints," in Software Engineering Conference (APSEC, 2013 20th Asia-Pacific, vol. 2. IEEE-CPS, Dec 2013, pp. 152–158.

[16] S. Chodrow, F. Jahanian, and M. Donner, "Run-time monitoring of real-time systems," in Real-Time Systems Symposium, 1991. Proceedings., Twelfth, dec 1991, pp. 74 –83.

[17] M. Saadatmand, A. Cicchetti, and M. Sjödin, "Design of adaptive security mechanisms for real-time embedded systems," in Proceedings of the 4th international conference on Engineering Secure Software and Systems, ser. ESSoS'12. Eindhoven, The Netherlands: Springer-Verlag, 2012, pp. 121–134.

[18] N. Walkinshaw, K. Bogdanov, S. Ali, and M. Holcombe, "Automated discovery of state transitions and their functions in source code," Journal of Software Testing, Verification & Reliability, vol. 18, no. 2, Jun. 2008, pp. 99–121.

[19] H. Yan, D. Garlan, B. Schmerl, J. Aldrich, and R. Kazman, "Discotect: A system for discovering architectures from running systems," in Proceedings of the 26th International Conference on Software Engineering, ser. ICSE '04. Washington, DC, USA: IEEE, 2004, pp. 470–479.

[20] D. Lo and S.-C. Khoo, "SMArTIC: Towards Building an Accurate, Robust and Scalable Specification Miner," in Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ser. SIGSOFT '06/FSE-14. New York, NY, USA: ACM, 2006, pp. 265–275.

[21] V. Dallmeier, C. Lindig, A. Wasylkowski, and A. Zeller, "Mining Object Behavior with ADABU," in Proceedings of the 2006 International Workshop on Dynamic Systems Analysis, ser. WODA '06. New York, NY, USA: ACM, 2006, pp. 17–24.

[22] D. Lorenzoli, L. Mariani, and M. Pezze, "Automatic generation of software behavioral models," in Software Engineering, 2008. ICSE '08. ACM/IEEE 30th International Conference on, May 2008, pp. 501–510.

[23] N. Kothari, T. Millstein, and R. Govindan, "Deriving state machines from tinyos programs using symbolic execution," in Information Processing in Sensor Networks, 2008. IPSN '08. International Conference on, April 2008, pp. 271–282.

[24] Microsoft Windows Workflow Foundation, http://msdn.microsoft.com/en-us/vstudio/jj684582.aspx, Accessed: August 2014.

[25] To use State Tracking with WorkflowApplication, http://wf.codeplex.com/wikipage?title=Tracking%20states%20with%20WorkflowApplication, Accessed: August 2014.

[26] R. Marinescu, M. Saadatmand, A. Bucaioni, C. Seceleanu, and P. Pettersson, "A model-based testing framework for automotive embedded systems," in 40th Euromicro Conference on Software Engineering and Advanced Applications. Verona, Italy: CPS-IEEE, August 2014.

[27] ITS-EASY post graduate industrial research school for embedded software and systems, http://www.mrtc.mdh.se/projects/itseasy/, Accessed: August 2014.

 ISBN: 978-1-61208-367-4

Functional Testing Criteria Applied in a Database Project

Dianne Dias Silva, Edmundo Sérgio Spoto, Leandro Luís Galdino de Oliveira
Instituto de Informática (INF)
Universidade Federal de Goiás (UFG)
Goiânia, Brazil
Emails: {diannesilva, edmundo, leandroluis}@inf.ufg.br

***Abstract*—This paper reports the application of the functional testing criteria for a database project in an IT company, aiming to explore the main features that exist in the project of the company. The paper presents a set of required elements based on the functional testing and the results of test cases analyzed in the project in question. The article also presents the criteria that were most effective in detecting faults in the organization of the database project.**

Keywords-Software Testing; Database Testing; Functional Software Testing; Functional Testing Criteria in a Database Project.

I. INTRODUCTION

Gradually, software has come to play a relevant activity in everyday society, so its reliability can not be ignored. The reliability criteria are specified by parameters of quality and as a result, sets of Software Quality Assurance (SQA) activities can be defined by supportting the software development process.

In this context, the activity of software testing should be started in parallel with the software design, requiring good planning, since the determination of testing criteria to be used, the definition of Test Cases (TC) and mass test data, come from this stage of the development cycle.

The testing criteria determine the Required Elements (RE), which must be tested because in general, the exhaustive testing is not feasible. Also, the RE are associated with testing techniques that explore different aspects and functionality of the software, relating to the functional technique (Equivalence Class Partitioning and Boundary Value Analysis), the statement structure of the project code belonging to the structural technique (Based-on-Control-Flow and Based-on-Data-Flow) and typical faults inserted into the software during its implementation caused by error based techniques (Error Seeding and Mutation Analysis) [5].

In addition, the growing usage of database applications in both small and large organizations, requires that relevant characteristics of database project, as cardinality, domain attributes, functional dependency, among others, are treated.

However, the techniques, strategies and tools for testing application database are scarce. Chays et al. [6] presented a number of important features of database project testing to be explored both in the development and the in the operation stages of the project.

The testing criteria for database projects used in this paper were chosen from the set of the testing criteria presented by Souza [10] and from the criteria used by the functional testing technique suggested by Carniello [1].

Although promising, the criteria proposed in these studies have not been validated in real database project. Thus, in this paper, we report an experiment using these criteria in a database project of an Information Technology (IT) development company.

The experiment helped to demonstrate the importance of these criteria in the improvement of the database project. The chosen criteria was shown to contribute to the detection of different types of faults in the analyzed database project, enhancing the quality of applications that use the database.

Besides testing the criteria in real application, we also investigated which criteria have higher chances to contribute to the detection of specific database project faults.

Thus, this study aimed to:

- Use the RE functional testing criteria (Equivalence Class Partitioning and Boundary Value Analysis) and also the criteria generated by Souza [10] through the restrictions of the relational model based schemes (Structural Relationships, Domain Attributes, Keys, Referential Integrity, Semantic Integrity and Functional Dependency) in a database project of an IT company;
- Build and run the corresponding TC and;
- Measure the strength of each criterion, emphasizing the importance in relation to the detection of faults in the project in question.

This paper is organized as follows. Section 2 presents the main concepts and terminology in the context of software testing for database project, as well as the criteria explored in this work. Section 3 shows a case study to be explored with the functional testing criteria on database project. Section 4 presents the results obtained. Section 5 presents the conclusion and future work.

II. BACKGROUND OF FUNCTIONAL TESTING IN A DATABASE PROJECT

The database testing techniques are applied during the creation of a database application, aiming to evaluate six levels of integrity which are: the Structural Relationships, the Domain Attributes, the Keys, the Referential Integrity, the Semantic Integrity and the Functional Dependency.

 ISBN: 978-1-61208-367-4

The functional testing criteria (Equivalence Class Partitioning and Boundary Value Analysis), have been adapted to test the attributes of the tables (Domain Attributes criterion) in order to check valid and invalid values on the domains used by the project, which would result in eight testing criteria for the database project.

The following are the definitions and terminology adopted for each criteria of the functional testing in a database project. In Section A, the functional testing criteria is presented before introducing the adaptations for their use in database projects.

A. Functional Testing Technique

The functional testing is a technique used in the creation of RE in order to exercise the values of the domain of each attribute of the tables in the database project [9][14]. Thus, the functional testing contributes to the improvement of the creation of the tables and also forces the creation of checks for each of the attribute types, in order to respect the database constraints. Functional testing can be done using the Data Manipulation Language (DML) or in conjunction with the database application software [5][10].

The functional testing technique contributes to detect faults that occur when specifying the boundaries of values that can be assigned to each attribute [13]. Moreover, an elaborate specification obtained by the user in the analysis phase is critical to identify these faults.

Therefore, the functional testing criteria are based on the database specification to generate the RE which are used to produce the corresponding TC. However, the generated TC must test the criteria without affecting other database constraints that are not related to the criteria being checked [1][11].

Then, it is understood that the database specification is used both to build a program and to contribute to the generation of RE-based on the specification criteria, and subsequently indicate mechanisms for the production of TC.

In the database project, presented in the case study, we used both the Equivalence Class Partitioning and Boundary Value Analysis as criteria to test tables attributes. These criteria are presented in the following two sections.

1. Equivalence Class Partitioning Criteria

The Equivalence Class Partitioning criteria is a black box testing technique that divides the input domain (of the attributes) through the specification conditions of a given data type classes, i.e., equivalence classes, of which TC are derivative [13].

Once the equivalence classes have been established, it can be assumed, with some certainty, that any member of a class can be considered a representative of it, and every member value should behave similarly, i.e., if one member causes a fault, then any other will also cause the same fault. Thus, the criteria reduces the input domain to a passive size to be treated during the testing activities [5].

An equivalence class represents a set of valid states (expected inputs) or invalid entries (not expected) to the entry conditions, here represented by the attributes of the database tables [7].

The usage of the equivalence classes is composed by two phases: identification of equivalence classes and the generation of the corresponding TC [8].

When an input attribute of an equivalence class results in [13]:

- *Use of Intervals*: One valid and two invalid class are defined, i.e., an invalid value would be well below the lower limit and well above the upper limit;
- *Use of Specific Value*: One valid and two invalid class are defined; i.e., the value (valid) itself and a lower value, and other higher (invalid);
- *Use of an Element of a Set*: A valid class (within the set) and an invalid (outside the set) are defined;
- *Use Boolean*: A valid class (T or F) and invalid one (other than T or F) are defined.

Thus, partitioning into equivalence classes for the attributes of the tables involved, aims to produce TC who discover several classes of errors and thereby reduce the total number of TC required to satisfy the criteria [8][13].

However, this criterion can also be classified as a systematic method for the assessment of requirements, in addition to restricting the number of existing TC [3][4].

And besides that, another black box testing technique called Boundary Values Analysis criteria uses the approaches of Equivalence Class Partitioning, being seen as complementary, thus making it more systematic [13].

2. Boundary Value Analysis Criteria

The Boundary Value Analysis criteria checks more rigorously the boundaries associated with the conditions of the input attributes, i.e., exercising the boundary values [5].

And according to Myers [8], it can be said that the TC, which explores the boundary conditions, has a higher probability of finding faults. This criterion exercise the conditions of entry, and also derived the TC output to the domain when necessary [7].

The guidelines for the Boundary Value Analysis are similar to Equivalence Class Partitioning criteria as the following [13]:

- If an input condition to specify an interval determined by the values *A* and *B*, TC must be designed with values *A* and *B*, just above and just below *A* and *B* respectively;
- If an input condition specify multiple values, TC are created to exercise minimum and maximum values. Values just below and just above the minimum and maximum are tested;
- Application to output conditions, the first and second guide;
- If the internal data structures of the program have identified limits, must be projected to TC to exercise this data structure at its boundary.

Finally, if a tester apply all these guidelines, the test itself, and is more systematically, it will be complete, having a greater likelihood of fault detecting.

B. Functional Testing Specific of the Database Projects

The functional testing in a database project is to validate the specification through the DML statements, which

 ISBN: 978-1-61208-367-4

contribute in detecting various problems in the construction of the database project, making this type of testing becomes difficult for the following reasons [2]:

- The construction of the database to test (choice of schemes and values) involves some important and relevant factors in the generation of TC, to meet each RE of the criteria. The selection of data is essential to getting a good set of TC, since it will be the entrance to any Structured Query Language (SQL) statements;
- The applications are not just a set of statements, in preparing the database testing. Therefore, the data should be useful to the greatest possible number of instructions, for loading test data with different information for each query has a high cost;
- The information generated for testing may be modified during the execution of SQL statements. Consequently, when designing a database to assist the test, it is necessary to consider the order in which SQL statements are executed and whether they will modify the data to be input for subsequent executions with a view that relations are persistent variables;
- As in imperative languages, SQL statements can be parameterized by variables and constants and when designing the test plan, these inputs should also be considered in addition to the test data provided by the same functional testing criteria of database project;
- The adequacy of the data test unit generated it is necessary to check whether the test unit really covers all possible situations and whether the output obtained by applying the test plan satisfies the requirements for which the database has been designed in such a case.

Anyway, the functional testing of the database project involves the following steps [6]:

- Extraction of information from database schema;
- Generation of test data and filling in the database testing;
- Generation of TC as input for database;
- Validation of the state of the database and exit after execution.

To exercise the test in database project, some criteria may be used, aiming to cover different fault types.

C. Functional Testing Criteria in a Database Project

The specific criteria for database have used features exercising relations represented in the database. Thus, some criteria require the generation of TC that exercise the attributes of the same relations and other criteria that exercise the attributes of different relations, which forces the production of TC that involves a number of DML statements in one or more connections to the test.

In this work, we use the term Relation instead of Table, to keep the terminology relational database. Taking into account that the functional testing approaches consider the domain variables based on the system specification in order to work out a database application and other characteristics of the database are investigated through based criteria in: the Structural Relationships, the Domain Attributes, the Keys, the Referential Integrity, the Semantic Integrity and the Functional Dependency [10].

1. Based-on-Structural-Relationships Criterion

The Based-on-Structural-Relationships criterion has two sub-criteria that exercise multiple relations simultaneously during the test: *all-maximum-cardinality* and *all-the-minimum-cardinality*. Given two Relations *A* and *B*, then the Based-on-Structural-Relationship criterion must generate TC that exercises the cardinality relationships between *A* and *B* in order to verify their specifications [10].

Definition 1: A test data set *T* satisfies the subcriteria *all-the-maximum-cardinality* constraints of the structural relationships criteria (one-to-one, one-to-many, many-to-one and many-to-many) between two Relations *A* and *B* if the actions of cardinality between *A* and *B* are met by application.

The sub-criteria *all-maximum-cardinality* is called exercised when *T* satisfy the criteria, ensuring that:

- Relation *A* has a one-to-one relationship with Relation *B* or;
- Relation *A* has a (zero or many)-to-one relationship with Relation *B* or;
- Relation *A* has a one-to-many relationship with Relation *B* or;
- Relation *A* has a many-to-many relationship with Relation *B*.

Definition 2: A test data set *T* satisfies the sub-criteria *all-minimum-cardinality* if the structural constraints of relationship (total and partial participation) are exercised between the Relation *A* and Relation *B* and if the actions of minimum cardinality between *A* and *B* are met by the database application.

The sub-criteria *all-minimum-cardinality* is considered to exercise when *T* satisfy the criteria, ensuring that there exists at least one relationship between *A* and *B*, total or partial.

2. Based-on-Domain-Attributes Criterion

Souza [10] defined the Based-on-Domain-Attributes criterion: *all-domain-attributes*, since is the same exercising the attributes of the same relation.

Definition: A test data set *T* satisfies the sub-criteria *all-domain-attributes* if all domain constraints (Check, Data Types and Allow Nulls) of the attributes of a relation are satisfied.

The sub-criteria *all-domain-attributes* of a relation is called exercised when *T* satisfies the criteria, ensuring that the values of the attributes domain of this relation:

- Whether checked all valid and invalid conditions for each attribute, respecting its data type;
- The conditions specified in accordance with Check (valid and invalid situations in relation to clause Check) clause were satisfied;
- Comply with conditions of null or not null values established by the Allow Nulls (null or not null) clause.

 ISBN: 978-1-61208-367-4

3. Based-on-Keys Criterion

In the Based-on-Keys criterion, defined by Souza [10], there is a need for exercising existing rules in Database Management Systems (DBMS) in which all Primary Keys (PK) must be unique and not null. The sub-criteria were established: *all-primary-keys.*

Definition: A test data set *T* satisfies the sub-criteria *all-primary-keys* if all restrictions related to PK of a relation are satisfied.

The sub-criteria *all-the-primary-keys* is called exercised when *T* satisfy the criteria, ensuring that:

- Occurring uniqueness of the value of PK;
- The PK value is not null.

4. Based-on-Referential-Integrity Criterion

The Based-on-Referential-Integrity criterion has as sub-criteria for the exercise of another relation: *all-foreign-keys*. This means that the references between the relations must satisfy the constraints between non-verbal relationships of two or more relations [10].

Definition: A test data set *T* satisfies the sub-criteria *all-foreign-keys* if the referential integrity constraints, Foreign Key (FK) and relationship between relations, or a relationship between *A* and *B* are satisfied.

The sub-criteria *all-foreign-keys* is called exercised when *T* satisfy the criteria, ensuring that:

- A tuple in Relation *A*, referenced by FK, belongs to other Relation *B* be the result of an existing tuple in the relationship between relations *A* and *B*;
- A set of attributes FK in the scheme of the Relation *A* is a FK of the relationship that references the Relation *B*;
- The attributes of FK of the Relation *A* have the same domain as the attributes of the PK of Relation *B*.

5. Based-on-Semantic-Integrity Criterion

Souza [10] defined the Based-on-Semantic-Integrity criterion: *all-semantic-attributes*. Being that it exercises the actions of semantic attributes and allowed values transitions valid values are in the same relation.

Definition: A test data set *T* satisfies the sub-criteria *all-semantic-attributes* if all semantic integrity constraints (between attributes and Check of dependent attributes) of a relationship are satisfied and the dependent attributes are in the same relation.

The sub-criteria *all-semantic-attributes* is called exercised when *T* to satisfy the criteria, ensuring that:

- The value attribute of a relation satisfies the semantic condition depending if the attribute may be the same relation or in a different relation.

The Semantic Integrity is presented here as the complement of Functional Dependency when it falls on the semantics of the attribute in question.

For example, a date of birth of a parent regarding the date of birth of a descendant or the salary of an employee should not exceed the salary of the manager of the employee.

6. Based-on-Functional-Dependency Criterion

The Based-on-Functional-Dependency criterion, defined by Souza [10], exercises the attributes distinct between the same relation or different relations to which it belongs. The sub-criteria was established: *all-attributes-functionally-dependent*.

Definition: A test data set *T* satisfies the sub-criteria *all-attributes-functionally-dependent* if the restriction of functional dependency between attributes of one or more relations is satisfied.

The sub-criteria *all-attributes-functionally-dependent* is called exercised when *T* satisfy the criteria, ensuring that an attribute of a:

- Relation *B* uniquely determines another attribute of Relation *A* and actions occur in distinct dependency relations;
- Relation can also be dependent on another attribute in the same relation. This can occur whenever there is information of an attribute that are formed by the values of other attributes.

III. CASE STUDY

In partnership with Laboratory of Quality Milk (LQL) belonging to the Food Research Center, Veterinary School of the Federal University of Goiás (Universidade Federal de Goiás), at Goiânia, was developed the Panel of Quality Milk (PQL) solution, which aims to provide customers the LQL a set of milk strategic information analyzed in the laboratory, plus a knowledge base produced by researchers at the institution [12].

The goal of this solution is to encourage continuous learning and the improvement of the final quality of the Brazilian milk, leading strategic real time information to the agents of the milk chain.

For dairy, the solution helps reducing operational costs, increasing profitability and opening new markets, promoting the improvement of the quality of the purchased milk yield and production.

Moreover, the operation of the PQL is provided by information extraction from milk samples (results of analysis) were collected and sent to the laboratory as well as those identifications its (producers, farms, animals and dairy products) directly from the LQL database.

The extraction of such data is performed daily at scheduled times, forming a database constituted by historical milk testing, which will be subject to statistical analyses by the PQL tool. These analyzes are presented to dairy through a website through authenticated access.

However, the integration architecture of the system is distributed in two locations: LQL (Database and Extractor) and PQL (Database, Integrator, Controllers and Web Browsers).

Finally, the application consists of:

- Registers (*Online Help*, *Cities*, *Farms*, *Dairy*, *Paper*, *People*, *Producers*, *Fixed Price Table*, *Bonus/Punishment Table*, *Errors Types*, *Users and Milk Volume*);

 ISBN: 978-1-61208-367-4

- Settings (*Fat X CCS*, *Histograms*, *Lactose X CCS*, *Protein X CCS*, *Industrial Performance*, *Tank Volume X CCS*, *Animal Volume X CCS* and *About System*);
- Panels grouped in versions: Basic (*Collection and Recollect with a Compliance IN62*), Standard (*History of Quality*, *Producer Mirror*, *Indicators of Routes* and *Route Mirror*), Advanced (*Decision Cube*, *Errors Cube*, *Distance and Volume* and *Distance Mirror*) and Full (*Analysis of Income*, *Statement of Producer*, *Pay Per Quality* and *View Cluster*).

A. Database of PQL Project

The DBMS employed in PQL Project was PostgreSQL was due to the fact that LQL make use of it and also the large data volume that the software will behave. Moreover, this DBMS is free, high performance, highly scalable platform.

The structure of this database includes tables, fifty eight, and eight of these were selected because they are essential and relate to virtually all other tables that make up the software. They are: *analysiserror*, *analysisresult*, *baseprice*, *farm*, *routefarm*, *monthclusterfarm*, *userroles* and *person*.

After this assignment, their relationships with other tables that make up the PQL database were identified and mapped, as the following:

- *analysiserror*: *animal*, *client*, *farm* and *sampleerror*;
- *analysisresult*: *animal*, *client*, *farm*, *sampleerror*, *casein*, *cbt*, *ccs*, *esd*, *est*, *fat*, *ibc*, *proteins* and *urea*;
- *baseprice*: *dairy*, *farm* and *price*;
- *farm*: *city*, *farmer*, *milkorigin*, *person1*, *person2* and *lqlcode*;
- *routefarm*: *dairy*, *farmcode* and *route*;
- *monthclusterfarm*: *farm* and *monthcluster*;
- *userroles*: *roles* and *users*;
- *person*: *city*.

They still used the Entity Relationship Diagram (ERD) to recognize these relationships, with the intention of presenting the dependencies between tables that have gone through database functional testing with their respective domains and specificities.

A testing technique with their respective criteria was established to derive their due RE provided the generation of TC and the extraction of its expected results.

Then, a specific and isolated environment testing was structured and also given a load on database project for the tables involved in this testing activity were populated.

Upon execution of the TC, a comparison between expected results and obtained results was performed aiming to verify the effectiveness of the criteria employed in the functional testing of the database project.

B. Exploited Criteria

Among the functional testing specific of the database project criteria presented in Section 2, not all were tested. For example, it was not possible to test only Based-on-Semantic-Integrity criterion, because this characteristic was not included in the Business Plan of PQL Project.

Some examples of RE, description of TC, Inputs and Expected Results applying these criteria, which were abstracted from the document TC project of the PQL project, i.e., the test specification of the same, are shown in Table I.

TABLE I. EXAMPLES OF FUNCTIONAL TESTING CRITERIA IN A DATABASE PROJECT

Functional Criteria	RE	TC	Input	Expected Results
Based-on-Structural-Relationships	Analyze the sample result of the animal.	Modify the date of the test result of an animal.	Analysis of results of animal "57" the date "2012-05-23" to "2012-07-02".	Occurrence
		Remove an animal that has a result of analysis.	All animals that have analysis results.	Not Occurrence
Based-on-Domain-Attributes (Equivalence Class Partitioning and Boundary Value Analysis)	Specify the creation date of a farm route.	Enter a valid date in the creation of a farm route.	Date = 2011-05-05.	Occurrence
		Enter a valid null in the creation of a farm route.	Date = *null*.	Not Occurrence
Based-on Keys	Check the consistency of the PK of a farm.	Insert a single PK on a farm.	PK = 943.	Occurrence
		Insert a null PK on a farm.	PK = *null*.	Not Occurrence
Based-on-Referential-Integrity	Analyze the sample error of the animal.	Insert a parsing error for a nonexistent animal.	Error Analysis "193" for the animal nonexistent "0".	Not Occurrence
		Remove an animal that has the error analysis.	Animal which has error analysis.	Not Occurrence
Based-on-Functional-Dependency	Determine the client's name and dairy.	Modify the name of the client and dairy.	Dairy = Parmalat Brazil S/A Food Industry.	Occurrence
		Modify the name of the client and dairy to a null value.	Dairy = *null*.	Not Occurrence

The Based-on-Domain-Attributes criterion was exercised in conjunction with Equivalence Class Partitioning and Boundary Value Analysis criteria because both evaluate the

specificities of the attributes that make up the database project tables.

Thus, the Based-on-Domain-Attributes criterion was exercised along with the functional technique being Equivalence Class Partitioning and Boundary Value Analysis criteria, because of them assessing the specificity of each of the attributes that make up a database table.

The functional testing criteria of the database project, explored in this paper are shown in Table II.

TABLE II. FUNCTIONAL TESTING CRITERIA IN A DATABASE PROJECT

Functional Criteria	Functional Subcriteria	TC Exercises
Based-on-Structural-Relationships	*all-maximum-cardinality* *all-minimum-cardinality*	Maximum Cardinality; Minimum Cardinality.
Based-on-Domain-Attributes (Equivalence Class Partitioning and Boundary Value Analysis)	*all-semantic-attributes*	Occurrence of a Domain; Allow Null Value.
Based-on-Keys	*all-primary-keys*	PK; Allow Null Value.
Based-on-Referential-Integrity	*all-foreign-keys*	FK; Permit Null Key.
Based-on-Functional-Dependency	*all-attributes-dependent-functionally*	Dependent Attribute.

Thus, other criteria database demonstrates aspects of verifying how it was built and even though the current DBMS preserve these properties, they were included only for verification.

According to the tables of the database and the established performance criteria yielded a model capable of revealing the RE needed to obtain their corresponding TC.

TABLE III. EXTRACTION MODEL OF THE FUNCTIONAL TESTING CRITERIA IN A DATABASE PROJECT

Functional Testing Criteria	
Equivalence Class Partitioning	
Condition	For each attribute of a Table, Attribute values with sequential domains: L_i until L_s.
Group 1	**RE01** – Valid value until L_s ranging from L_i (L_i <= Attribute <= L_s); **RE02** – Invalid value below the L (Attribute < L_i); **RE03** – Invalid value higher than L_i (Attribute > L_s).
Condition	For each attribute a table attribute belonging to a set of values: Attribute ∈ {a, b, c, d}.
Group 2	**RE04** – Valid within the set value (Attribute ∈ {a, b, c, d}); **RE05** – Invalid value out of the set (Attribute ∉ the set {a, b, c, d}: {e}).
Boundary Value Analysis	
Condition	For each attribute of a table, with attribute values in the domain limit L.
Group 3	**RE06** – Valid value equal to the limit of **L** (**L** = Attribute); **RE07** – Invalid value lower next to **L** (Attribute < **L**); **RE08** – Invalid value near the top **L** (Attribute > **L**).
Based-on-Domain-Attributes	
Condition	For each attribute of a table, the field mapping: *Data Type* and *Allow Nulls*.
Group 4	**RE09** – Data Type (Attribute of type *Numeric*); **RE10** – Data type (Attribute of type *Date/Time*); **RE11** – Data type (Attribute of type *String*); **RE12** – Allow Nulls (Attribute *Null*); **RE13** – Allow Nulls (Attribute *Not Null*).
Based-on-Keys	
Condition	For each key of a table, the mappings of keys: *PK*.
Group 5	**RE14** – PK (Candidate Key Simple); **RE15** – PK (Candidate Key Composite).
Based-on-Structural-Relationship	
Condition	For each ratio of a table, the mappings of relations: *Relationship*, *Cardinality* and *Dependence*.
Group 6	**RE16** – Relations (Relations Association and Dependence); **RE17** – Cardinality (Relationship of Cardinality 1 – 1); **RE18** – Cardinality (Relationship of Cardinality 1 – N); **RE19** – Cardinality (Relationship of Cardinality N – N); **RE20** – Dependence (Specialization); **RE21** – Dependence (Generalization);
Based-on-Referencial-Integrity	
Condition	For each key of a table, the mappings of keys: *FK*.
Group 7	**RE22** – FK (Relationship cardinality); **RE23** – FK (Dependence).
Based-on-Functional-Dependency	
Condition	For each attribute of a table, the field mapping: *Check*.
Group 8	**RE24** – Check (Extend Relationship).

Therefore, the organization of these criteria is as shown in Table III, considering the specificities of both functional testing criteria as the for database criteria.

IV. OBTAINED RESULTS

Results for functional testing criteria in a database project used in this study were obtained through test analysis based on the coverage percentage for the quantity of RE exercised by the TC.

Altogether, there were 443 RE, generating 425 TC is needed in this step. Therefore, it was found that all the TC has been run and the database project also acquired that is a 100% coverage for the criteria.

Still, were achieved the results of the functional testing criteria (Equivalence Class Partitioning and Boundary Value Analysis) employees in Based-on-Domain-Attributes criterion. Therefore, all TC related in the criteria were executed and, furthermore, achieved a 100% coverage.

The results stemmed from the implementation of a specific functional testing in a database project through the exercise of the analyzed criteria to be presented in Table IV.

TABLE IV. RESULTS OF THE TEST RUN

Functional Testing Criteria	RE	TC	Defects
Based-on-Domain-Attributes (Equivalence Class Partitioning and Boundary Value Analysis)	171	171	9
Based-on-Keys	28	21	0
Based-on-Structural-Relationships	124	113	0
Based-on-Referential-Integrity	100	100	0
Based-on-Functional-Dependency	20	20	0
Grand Total	**443**	**425**	**9**

In general, it is observed that the other specific criteria of database testing help verify that the project meets specified correctly to ensure a good quality of the generated information.

The types of fields *date*, *number*, *text*, *email* and *website* were verified through Based-on-Domain-Attributes criterion along with functional testing criteria Equivalence Class

 ISBN: 978-1-61208-367-4

Partitioning and Boundary Value Analysis, which consequently showed the faults in a database project.

Finally, the Boundary Value Analysis criteria detected most of the faults identified in the database project because a large amount of these faults is in the limits of the domains of its attributes.

V. CONCLUSION AND FUTURE WORK

According to functional testing criteria on a database project, there was a high efficiency degree in detection of faults during the execution of TC for the RE generated in PQL's Reports. All functional testing criteria applied reached a coverage of 100% in relation the RE.

Nine faults were detected over the following criteria, Equivalence Class Partitioning (three faults) and Boundary Value Analysis (six faults) together with Based-on-Domain-Attributes criterion.

The Equivalence Class Partitioning criteria contributed to the definition of TC by the class of faults, reducing the total number of TC generated in the PQL's Project. The Boundary Value Analysis criteria allowed to observe most faults tend to occur at the borders of the domain.

Furthermore, because the Based-on-Keys, Based-on-Structural-Relationships, Based-on-Referential-Integrity and Based-on-Functional-Dependency criterion contributing with corrections in existing restrictions on a project database, so no fault was detected during the execution of the TC. The Based-on-Semantic-Integrity criterion was not used in this project considering that the documentation does not make any reference to these dependencies.

Finally, the combination of Based-on-Domain-Attributes criterion also provides Equivalence Class Partitioning and Boundary Value Analysis criteria by exploring a high level of faults detection, just treating the specific attributes. And thus, can be utilized in the database project.

For future work, an object of study with the purpose of applying the criteria presented is being constructed, such as the Based-on-Functional-Dependency and Based-Semantic-Integrity. These criteria can improve the detection of failures a project database.

REFERENCES

[1] A. Carniello, Test Structure Based on Use-Case, FEEC/UNICAMP, Campinas, 2003.

[2] A. D. Suarez, A. S. Simão, J. C. Maldonado, and P. C. Masiero, "Using an SQL Coverage Measurement for Testing Database Applications", In: ACM SIGSOFT Software Engineering Notes, New York, pp. 253-262, 2004.

[3] A. L. Domingues, Assessment Criteria and Test Tools for OO Programs, ICMC/USP, São Carlos, 2002.

[4] A. M. Vincenzi, E. F. Barbosa, J. C. Maldonado, M. E. Delamaro, M. Jino, and S. R. S. Souza, Introduction to Software Testing, Teaching Notes, ICMC-USP, São Carlos, 2004.

[5] A. M. Vincenzi, J. C. Maldonado, and S. C. Fabbri, Introduction to Software Testing: Functional Testing, Rio de Janeiro: Elsevier, 2007.

[6] D. Chays, E. J. Weyuker, F. I. Vokolos, P. G. Frankl, and S. Dan, "A Framework for Testing Database Applications", In Proc. of the ACM SIGSOFT Intl. Symp. On Software Testing and Analysis, Vol. 25 Issue 5, August 2000, pp. 49-59.

[7] G. J. Myers, Software Reliability Principles and Practices, 1st ed. New York: John Wiley & Sons, INC., p. 360, 1976.

[8] G. J. Myers, T. Badgett, and T. M. Thomas, The Art of Software Testing. 2nd ed., New York: John Wiley & Sons, INC., 2004.

[9] I. Burnstein, Pratical Software Testing: A Process Oriented Approach, New York: Springer-Verlag, p. 709, 2002.

[10] J. P. Souza, Functional Testing Application DB Based on UML diagram, UNIVEM, Marília, 2008.

[11] J. Tian, Software Quality Engineering, Texas: John Wiley & Sons, INC., p. 412, 2005.

[12] Milk Panel, http://www.paineldoleite.com.br/site/ Aug/Sept 2014.

[13] . S. Pressman, Software Engineering, 6th ed., São Paulo: McGraw-Hill, p. 720, 2006.

[14] W. E. Lewis, Software Testing and Continuous Quality Improvement, 2nd ed., Florida: CRC Press LLC, p. 534, 2004.

 ISBN: 978-1-61208-367-4

Automatic Unit Test Generation and Execution for JavaScript Program through Symbolic Execution

Hideo Tanida, Tadahiro Uehara
Software Engineering Laboratory
Fujitsu Laboratories Ltd.
Kawasaki, Japan
Email: {tanida.hideo,uehara.tadahiro}
@jp.fujitsu.com

Guodong Li, Indradeep Ghosh
Software Systems Innovation Group
Fujitsu Laboratories of America, Inc.
Sunnyvale, CA, USA
Email: {gli,indradeep.ghosh}
@us.fujitsu.com

***Abstract*—JavaScript is expected to be a programming language of even wider use, considering demands for more interactive web/mobile applications. While reliability of JavaScript code will be of more importance, testing techniques for the language remain insufficient compared to other languages. We propose a technique to automatically generate high-coverage unit tests for JavaScript code. The technique makes use of symbolic execution engine for JavaScript code, and stub/driver generation engine which automatically generate stub for code of uninterest. Our methodology allows for fully automatic generation of input data for unit testing of JavaScript code with high coverage, which ensures quality of target code with reduced effort.**

Keywords–JavaScript, test generation, symbolic execution, stub generation.

I. INTRODUCTION

Extensive testing is required to implement reliable software. However, current industrial practice rely on manually-written tests, which result in large amount of effort required to ensure quality of final products or defects from inadequate testing.

Verification and test generation techniques based on formal approaches are considered to be solutions for the problem. One such technique is test generation through symbolic execution, which achieves higher code coverage compared to random testing [1]–[6].

In order to symbolically execute a program, input variables to the program are handled as symbolic variables with their concrete values unknown. During execution of the program, constraints to be met by values of variables in each execution path are obtained. After obtaining constraints for all the paths within the program, concrete values of input variables to execute every paths can be obtained, by feeding a solver such as Satisfiability Modulo Theory (SMT) [7] solver with the constraints. Normal concrete execution of the program using all the obtained data, results in all the path within the program went through.

Manually-crafted test inputs require effort for creation, while they do not guarantee running all the execution path in the target program. In contrast, test generation based on symbolic execution automatically obtains inputs to execute all the path within the program. As the consequence, it may find corner-case bugs missed with insufficient testing.

There are tools for symbolic execution of program code, including those targeting code in C/C++ [1][2][4], Java [3], and binary code [5][6]. It is reported that the tools can automatically detect corner-case bugs, or generate test inputs to achieve high code coverage.

Existing tools for JavaScript code include Kudzu [8] and Jalangi [9]. Kudzu automatically generates input data for program functions, with the aim of automatically discovering security problems in the target. Jalangi allows modification of path constraints under normal concrete executions, in order to obtain results different from previous runs. However, the tools could not be applied to unit testing of JavaScript code in field, due to limitations in string constraint handling and need for manual creation of driver/stub used for testing.

We propose a technique to generate test inputs for JavaScript code through symbolic execution on a tool SymJS. Test inputs generated by the tool allows for automatic unit test execution. After augmenting generated test inputs with user-supplied invariants, application behavior conformance under diverse context can be checked in a fully automatic fashion. Our proposal includes automatic generation of symbolic stubs and drivers, which reduces need for manual coding. Therefore, our technique allows for fully automatic generation of input data used in unit testing of JavaScript code. Test inputs generated by our technique exercise feasible execution paths in the target to achieve high coverage.

Our methodology has the following advantages to existing works. Our JavaScript symbolic execution engine SymJS is applicable to JavaScript development in field for the following reasons. First, our constraint solver PASS [10] allows test generation for programs with various complex string manipulations. Secondly, SymJS does not require any modification to the target code, while the existing symbolic executors for JavaScript [8][9] needed modifications and multiple runs.

Further, our automatic stub/driver code generation allows for fully automatic test data generation. An existing work [9] could be employed for generation of unit tests. However, it required manual coding of stub/driver, which requires knowledge

 ISBN: 978-1-61208-367-4

```
function func0(s,a) {
  if("".equals(s)) { // block 0
    s = null;
  } else {
    if(s.length <= 5) { // block 1
      a = a + status;
    } else {
      if("".equals(s)) { // block 2
        Lib.m0(); // Unreachable
      } else { // block 3
        Lib.m1();
      }
    }
  }
  if(a <= Lib.m2()) { // block A
    a = 0;
  } else { // block B
    a = a + s.length; // Error with null s
  }
}
```

Figure 1. Code Fragment Used to Explain our Methodology: s, a, Lib.m2() may Take Any Value

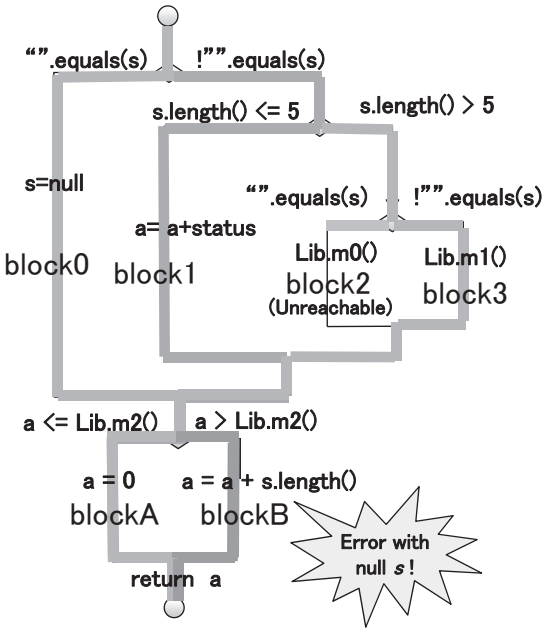


Figure 2. Execution Paths within Code Shown in Figure 1

on symbolic execution and error-prone. Our fully automatic technique can be applied to development in practice.

The rest of this paper is organized as follows. Section II explains the need for automatic test generation/execution with an example, and introduces our test input generation technique through symbolic execution. Section III describes our method to automatically generate stub/driver code used in test generation/execution. Evaluation in Section IV shows applicability and effectiveness of our technique. Finally, we come to the conclusion in Section V and discuss future research directions.

II. Background and Proposed Test Generation Technique

A. Demands for Automatic Test Generation

Generally, if a certain execution path in a program is exercised or not, depends on input fed to the program. Therefore, we need to carefully provide sufficient number of appropriate test input data, in order to achieve high code coverage during testing.

For example, function `func0()` shown in Figure 1 contains multiple execution path. Further, whether each path is exercised or not depends on input fed to the program, which are value of arguments `s,a` and return value of function `Lib.m2()`. Current industrial testing practice depends on human labor to provide the inputs. However, preparing test inputs to cover every path within software under test requires large amount of efforts. Further, manually-created test inputs might not be sufficient to exercise every path within the target program.

Figure 2 shows possible execution path within the example in Figure 1. In the example, there are two set of code blocks and whether blocks are executed or not depend on branch decisions. The first set of the blocks contains blocks 0-3, and the second set contains blocks A-B. Conditions for the blocks to be executed are shown at the top of each block in Figure 2. Block 2 has a contradiction between conditions for execution and will never be executed. However, the other blocks have no such contradiction and executable. Tests to execute every possible combination of blocks 0,1,3 and blocks A,B correspond to $3 \times 2 = 6$ set of values for the inputs.

TABLE I. Constraints to Execute Paths in Figure 2 and Satisfying Test Inputs (under assumption status=-1)

Test No.	Blocks Executed	Path Conditions	Test Data
1	0,A	`"".equals(s)` ∧ `a<=Lib.m2()`	`s=""`, `a=0` `Lib.m2()=0`
2	0,B	`"".equals(s)` ∧ `a>Lib.m2()`	`s=""`, `a=0` `Lib.m2()=-1`
3	1,A	`!"".equals(s)` ∧ `s.length <= 5` ∧ `a-1<=Lib.m2()`	`s="a"`, `a=0` `Lib.m2()=0`
4	1,B	`!"".equals(s)` ∧ `s.length <= 5` ∧ `a-1>Lib.m2()`	`s="a"`, `a=1` `Lib.m2()=0`
5	3,A	`!"".equals(s)` ∧ `s.length > 5` ∧ `a<=Lib.m2()`	`s="aaaaaa"`, `a=0` `Lib.m2()=0`
6	3,B	`!"".equals(s)` ∧ `s.length > 5` ∧ `a>Lib.m2()`	`s="aaaaaa"`, `a=0` `Lib.m2()=-1`

TABLE I shows combinations of blocks to execute and path condition to be met by arguments `s,a` and return value of `Lib.m2()`. In the example, it is possible to obtain concrete values meeting the conditions for the inputs, and the values can be used as test inputs. We will discuss how to automatically obtain such test inputs in the sequel.

B. Test Input Generation through Symbolic Execution

We propose a methodology to automatically generate test inputs with SymJS, a symbolic execution engine for JavaScript. During symbolic execution of a program, constraints to be met in order to execute each path (shown as "Path Conditions" in

 ISBN: 978-1-61208-367-4

TABLE II. INSTRUCTIONS WITH THEIR INTERPRETATIONS MODIFIED FROM ORIGINAL RHINO

Arithemetic/Logical Operations	ADD, SUB, MUL, DIV, MOD, NEG, POS, BITNOT, BITAND, BITOR, BITXOR, LSH, RSH, URSH etc.
Comparisons	EQ, NE, GE, GT, LE, LT, NOT, SHEQ, SHNE etc.
Branches	IFEQ, IFNE, IFEQ_POP etc.
Function Calls	RETURN, CALL, TAIL_CALL etc.
Object Manipulations	NEW, REF, IN, INSTANCEOF, TYPEOF, GETNAME, SETNAME, NAME etc.
Object Accesses	GETPROP, SETPROP, DELPROP, GETELEM, SETELEM, GETREF, SETREF etc.

TABLE I) are calculated iteratively. After visiting every possible path within the program, constraints for all the path are obtained. Concrete values of variables meeting the constraints can be obtained with solvers such as SMT solver. Obtained values are input data to exercise paths corresponding to the constraints, which we can use for testing.

While JavaScript functions are often executed in a event-driven and asynchronous fashion, our technique focuses on generation of tests which invoke functions in deterministic and synchronous orders. We assume the behavior of generated tests are reasonable, considering what is inspected in current JavaScript unit tests in field, as opposed to integration/system testing. Each generated test data exercise single path within the target, and only single data is generated for each path.

SymJS allows for symbolic execution of JavaScript code. SymJS interprets bytecode for the target program, and symbolically executes it in a way KLEE [2] and Symbolic JPF [3] do. SymJS handles program code meeting the language standard defined in ECMAScript [11].

SymJS is an extended version of Rhino [12], an open-source implementation of JavaScript. Our extensions include symbolic execution of target code, constraint solving to obtain concrete test input data, and state management. While there are existing symbolic executors for JavaScript, SymJS does not reuse any of their code base. TABLE III shows comparison between SymJS and existing symbolic executors.

SymJS interprets bytecode compiled from target program source code. This approach is taken by existing symbolic executors such as KLEE [2] and Symbolic PathFinder [3]. Handling bytecode instead of source allows for implementation of symbolic executors without dealing with complex syntax of the language. SymJS is implemented as an interpreter of Rhino bytecode, which updates the program state (content of heap/stack and path condition) on execution of every bytecode instruction. Upon hitting branch instruction, it duplicates the program state and continues with the execution of both the branches.

In order to implement symbolic execution of target programs, we have modified interpretation of the instructions shown in TABLE II from the original Rhino. Handling of instructions for stack manipulation, exception handling, and variable scope management remain intact.

For example, an instruction ADD op1 op2 is interpreted as follows. (1) Operands op1 and op2 are popped from stack. The operands may take either symbolic or concrete value. (2) Types of the operands are checked. If both the operands are String, the result of computation is the concatenation of the operands.

TABLE III. COMPARISON OF SYMBOLIC EXECUTORS

Tool	Target Lang.	Sym. VM	Dep./Cache Solving	String Solving
SymJS	JavaScript	Yes	Yes	Yes
KLEE [2]	C	Yes	Yes	No
SAGE [6]	x86 binary	No	Yes	No
Sym JPF [3]	Java	Yes	No	No
Kudzu [8]	JavaScript	No	No	Yes
Jalangi [9]	JavaScript	No	No	Limited

TABLE IV. REPRESENTATION OF STATES IN FUZZING AFTER EXECUTING CODE ON FIGURE 1 UNDER PATH CONDITIONS IN TABLE I

Test No.	Blocks Executed	State Representation
1	0,A	L;L
2	0,B	L;R
3	1,A	R;L;L
4	1,B	R;L;R
5	3,A	R;R;R;L
6	3,B	R;R;R;R

If they are Numeric, the result is the sum of the operands. Otherwise, values are converted according to ECMAScript language standard, and the result is either concatenation or addition of the obtained values.

Comparison instructions are followed by branch instructions in Rhino bytecode. SymJS handles comparison and branch instruction pairs as in the following. First, it creates Boolean formula corresponding to result of comparison after necessary type conversions. Assuming the created formula is denoted by symbol c, we check if c and its negation $\neg c$ are satisfiable together with path condition pc. In other words, we check for satisfiablity of $pc \wedge c$ and $pc \wedge \neg c$. If both are satisfiable, we build states s_1, s_2 corresponding to $pc \wedge c$ and $pc \wedge \neg c$ and continue with execution from states s_1 and s_2. If one of them is satisfiable, the state corresponding to the satisfiable one is chosen and execution resumes from that point.

SymJS supports two ways to manage states which are created on hitting branches etc. The first method is to store program state variables including content of heap/stack, as is done in [2][3]. The second method is to remember only which side is taken on branches. This method needs to re-execute the target program from its initial state on backtracking. However, it benefits from its simple implementation and smaller memory footprint. The method is called "Fuzzing" and similar to the technique introduced in [4][6]. However, our technique is implemented upon our symbolic executor and does not need modification of target code required with the existing tools [8][9] for JavaScript.

During symbolic execution of programs through fuzzing, states are represented and stored only by which side is taken on branches. The information can be used to re-execute the program from its initial state and explore the state space target may take. States after symbolically executing the target program in Figure 1 with path conditions corresponding to tests 1-6 in TABLE I, are represented as shown in TABLE IV during fuzzing. Symbols *L,R* denote left/right branch is taken on a branching instruction.

For each of state representations shown in TABLE IV, corresponding path condition can be obtained. TABLE I includes path conditions for the states in TABLE IV. If it is possible to obtain solutions satisfying the constraints, they can

 ISBN: 978-1-61208-367-4

be used as inputs used during testing. Constraints on numbers can be solved by feeding them into SMT solvers. However, SMT solvers cannot handle constraints of strings, which is heavily used in most of JavaScript code. Therefore, we employ constraint solver PASS [10] during test input generation.

PASS can handle constraints over integers, bit-vectors, floating-point numbers, and strings. While previous constraint solvers supporting string constraints used bit-vectors or automata, PASS introduced modeling through parameterized-arrays which allows for more efficient solving. As the consequence, it can solve most of constraints corresponding to string manipulations within ECMAScript standard.

C. Symbolic Stubs and Drivers

Symbolic variables are targets of test input generation through symbolic execution. SymJS allows definition of symbolic variables through function calls. The code snippet below shows an example of defining symbolic string variable.

```
var s = symjs_mk_symbolic_string();
```

While the example defines a symbolic variable of string type, functions `symjs_mk_symbolic_int()`, `symjs_mk_symbolic_bool()` and `symjs_mk_symbolic_real()` allow definition of symbolic variables with their type being integer, Boolean, and floating-point, respectively. While SymJS allow only string, integer, Boolean, and floating-point numbers to be symbolic, their constraints are retained on assignments/references as members of more complex objects, allowing generation of tests with value of object members changing.

In order to determine test inputs for the function `func0()` in Figure 1, additional code fragments are required. First, a symbolic driver shown in Figure 3 is required. The driver declares symbolic variables and passes them to the function as arguments. Stubs to inject dependencies are also required. A symbolic stub in Figure 4 includes a symbolic variable declaration. With the stub, the return value of the function call to `Lib.m2()` is included to test inputs obtained by SymJS.

Functions `symjs_mk_symbolic_*()` used to define symbolic variables is interpreted as expressions to define new symbolic variables during test generation. SymJS allows for normal concrete execution with the generated test inputs. During concrete execution, the functions return concrete values contained in test inputs. SymJS can export test inputs into external files in JavaScript Object Notation (JSON) format. The files can be read by test playback library which returns corresponding test input data on `symjs_mk_symbolic_*()` function calls. The library loaded into typical web browser enables execution of generated tests with no custom JavaScript interpreter.

III. Automatic Generation of Symbolic Stubs and Drivers

As explained in Section II-C, symbolic stubs and drivers are required to symbolically execute target functions and obtain test inputs. Symbolic stubs which return symbolic variables are used to generate return values of functions which are called from functions under test. Symbolic drivers are needed to vary arguments passed to functions tested.

While it is possible to employ manually implemented symbolic stubs and drivers, additional cost is required for implementation. Therefore, it is desirable to have symbolic stubs and drivers be automatically generated. Hence, we have decided to generate symbolic stubs and drivers in an automatic manner, and use them for test generation and execution.

A. Strategy for Generating Symbolic Stubs and Drivers

Our symbolic stub generation technique produces stub for functions and classes specified. Our driver generation technique emits code which calls functions specified.

As for stub generation, we have decided to generate functions which just create and return objects according to type of return value expected by caller. The following is the mapping between expected type and returned object:

- String, integer, Boolean and floating-point numbers which SymJS can handle as symbolic
  (Hereafter referred to as SymJS primitives):
  Newly defined symbolic variable of the corresponding type.
- Other classes:
  Newly instantiated object of the expected type. If the class is targeted for stub generation, newly instantiated stub object is returned.
- Void: Nothing is returned.

In order to create stubs for classes, stubs for constructors also need to be generated. Here, we generate empty constructors, which result in all stateless objects. Our approach assumes there is no direct access to fields of stub classes, and does not generate stubs for fields.

We have to note even in case type of return value is a non-SymJS primitive, we may get multiple test inputs. That is the case if functions defined in the returned object return symbolic variables. The situation happens if the non-SymJS primitive class contain functions which return objects of SymJS primitive class, and the non-SymJS primitive class is stubbed. Therefore, it is possible to obtain more than one set of test inputs by calling functions returning non-SymJS primitive.

Symbolic drivers generated with our technique have the following functionality:

- If the function to be tested is not static and needs an object instance to be executed, instantiate an object of the corresponding class and call the function
- If the function is a static one, just call the function

As arguments passed to the function, drivers give the following objects according to the expected types:

- SymJS primitives:
  Newly defined symbolic variable of corresponding type.
- Other classes:
  Newly instantiated object of the expected type. If the class is targeted for stub generation, newly instantiated stub object is passed.

 ISBN: 978-1-61208-367-4

```
var s = symjs_mk_symbolic_string();
var a = symjs_mk_symbolic_float();
func0(s,a);
```

Figure 3. Symbolic Driver to Execute Code in Figure 1

```
Lib.m2 = function() {
  return symjs_mk_symbolic_float();
};
```

Figure 4. Symbolic Stub Providing Lib.m2() Used in Figure 1

```
/** @return {Number} m2 value */
Lib.m2 = function() { ... };
```

Figure 5. Function Definition with an Annotation to Automatically Generate Symbolic Stub in Figure 4

```
/** @return {tx.Data} data */
tx.Ui.getValue = function(){ ... };
```

⇓

```
tx.Ui.getValue = function(){
    return new tx.Data();
};
```

Figure 6. Function with an Annotation Returning non-SymJS Primitive and Generated Symbolic Stub

```
/** @param {String} s
  * @param {Number} a */
function func0(s,a) { ... }
```

Figure 7. Annotations for Function under Test to Automatically Generate Symbolic Driver in Figure 3

The manner to choose arguments is similar to the one resolves what to return in symbolic stubs.

B. Generating Symbolic Stubs and Drivers from Annotations

Symbolic stub/driver generation strategy proposed in Section III-A requires type information from target code. Types of return values expected by caller are required for stub generation. Types of arguments passed to functions under test are required to generate drivers.

However, JavaScript is a dynamically typing language which makes it difficult to determine type of return values and arguments prior to run time. On the contrary, many JavaScript programs have some expectations in types of return values and arguments, which are often given in Application Programming Interface (API) etc. Further, there is a way to express type information for JavaScript code in a machine readable manner, which is JSDoc-style annotation. Therefore, we have decided to obtain type information from annotations in JSDoc3 [13] convention, and generate symbolic drivers and stubs.

Symbolic stubs are generated from original source code of functions to generate stubs for. Functions need to contain annotations, which provide type information on return values of functions. Symbolic stub for a function can be generated if type of its return values is obtained from annotations.

JSDoc3 allows declaration of return value type, mainly through `@return` annotations. In order to generate symbolic stub for function `Lib.m2()` used in code snippet on Figure 1, an annotation like the one shown in Figure 5 is required. If such annotation is attached to original source code of the function, it is possible to figure out type for return values. From the obtained type for return values, the symbolic stub in Figure 4 can be generated in a fully automatic manner. The example demonstrates generation of symbolic stub for a function returning a SymJS primitive. An example of generating symbolic stub for a function which returns a non-SymJS primitive is shown in Figure 6.

Symbolic drivers are generated from source code of functions to be tested. Source code need to contain annotations expressing type of arguments passed to the function, in order to automatically generate symbolic driver to invoke the function.

Type of parameters passed to functions are often given with `@param` annotation for JSDoc3. Symbolic driver for the function `func0()` can be generated from the annotations in Figure 7, attached to the function. The annotations give types of parameters for the function, allowing generation of the symbolic driver in Figure 3.

The proposed technique for automatic generation of symbolic stub and drivers is implemented as plugins for JSDoc3. JSDoc3 allows implementation of custom plugins, and they may contain hooks to be invoked on finding classes or functions. Within the hooks, it is possible to obtain types for return values and parameters. The developed plugins automatically generate symbolic stubs and drivers for classes and functions contained in program source code fed to JSDoc3.

While we have proposed a technique to automatically generate symbolic stubs and drivers based on type information obtained from annotations in program, it is also possible to use type information from other sources. Such sources of type information include API specification documents.

IV. Evaluation

In order to confirm that our proposed technique can automatically generate and execute unit tests achieving high code coverage, we have performed experiments using an industrial JavaScript program. The program corresponds to the client part of web application implemented upon our custom framework for web application implementation. The program calls to API not defined in ECMAScript standard wrapped in our framework, and it contains only calls to standard API or our framework. We have to note common API to manipulate HTML Document Object Model (DOM) or communicate with servers are not part of ECMAScript standard and they are not used directly in the program. TABLE V shows statistics on the target program.

A. Generation of Symbolic Stubs and Drivers

In order to perform automatic generation of test input proposed in Section II, we have generated symbolic stubs and drivers with through technique explained in Section III.

Symbolic stubs are generated from source code of the framework used to implement the application. Source code has annotations meeting JSDoc3 standard which allow for retrieval

 ISBN: 978-1-61208-367-4

TABLE V. STATISTICS ON THE TARGET PROGRAM

#Line	#Function	#File
431	23	1

TABLE VI. STATISTICS ON THE FRAMEWORK SOURCE CODE USED FOR STUB GENERATION AND GENERATED STUB

#Line(Orig.)	#Line(Stub)	#Function	#File
2843	1304	154	13

of types for return values of functions. Stubs are sucessfully generated for all the classes and functions defined in the framework. TABLE VI lists figures on the original framework source code and generated symbolic stubs. As the program is implemented only upon API defined in ECMAScript language standard and the framework, all the stubs required for symbolic execution of the program are ready at this stage.

Symbolic drivers are generated from source code for the program under test. JSDoc3-style annotations can be found in the target program as well, and it is possible to determine types of arguments required for generation of symbolic drivers. Drivers for all 23 functions within the program are generated.

B. Test Input Generation and Test Execution

Each function within the target program is executed in a symbolic manner, using the automatically generated drivers and stubs. Test inputs containing concrete values of symbolic variables are obtained at the end of executions.

Symbolic execution of all functions finished *within 1 second* and test inputs are generated. Number of test inputs, which are assignments of concrete values into symbolic variables, differed between target functions. Only *1* test input is generated for functions with no branch, while number of tests varied to *27* obtained with a more complex function.

Target functions are concretely executed with test inputs obtained. Test playback library running on a web browser is used to replay the tests. Code coverage during testing is measured with JSCover [14], and line coverage of *92%* was obtained. The result shows our technique can generate unit test input achieving high code coverage fully automatically.

C. Code Not Covered in the Experiments

While the experimental results show that the proposed method can generate test input achieving high code coverage, 100% coverage is not reached, implying some portion of the target program is not exercised. The followings are the classes of code not executed with our methodology.

Code handling objects of unexpected type is not covered. As JavaScript is a dynamically typing language, objects of unexpected type might be returned by functions. In order to handle such scenario, the target program contained type checking and subsequent error handling code. However, symbolic stubs generated through our technique, always return a object of type described in source code annotation. Such stubs fail to utilize code portions handling objects of type different from annotations.

Code with no premise on object type is also missed. The target program contained code fragments which determine type of objects at run time and process them accordingly. However, our technique cannot cover such procedures. From functions with types of their return values unknown, we generate stubs returning default JavaScript “Object”. Therefore, code interacting with objects of custom class is uncovered.

Catch blocks handling exceptions is left. The target program contained catch blocks for exceptions thrown from the framework used in the program. However, after replacing the framework with the automatically generated symbolic stubs which throw no exception, they are not exercised.

V. CONCLUSIONS AND FUTURE WORK

A. Conclusions

We proposed a technique to automatically generate unit test input data for JavaScript code. The technique makes use of a symbolic execution engine, in order to achieve high code coverage during testing. The technique is a two-phase approach, consisting of the following fully-automatic steps:

1) Symbolic stub/driver generation based on type information obtained from annotations
2) Test input generation through symbolic execution of target code

The experiment with an industrial JavaScript program shows the technique can generate tests achieving line coverage of 92%. The result shows our technique can automate generation and execution of unit tests for JavaScript code.

B. Future Work

Future work includes more verification trials with variety of target programs. While we have performed experiments with programs of relatively small size, experiments on larger targets are also required.

In order to exercise target code missed in the experiment, symbolic stubs need to be improved. Code handling objects of unexpected/unknown type can be kicked by symbolic stubs which return various types of objects. Code handling exceptions can be triggered with symbolic stubs throwing exceptions. In addition to more complex automatic stub generation strategies, manual modifications to automatically generated stubs are considered effective to increase coverage.

In the experiment, we have targeted JavaScript code with HTML DOM handling encapsulated in our framework, allowing test generation and execution only with creation of symbolic stub for the framework. In order to target JavaScript code containing manipulation on HTML DOM, symbolic stubs for HTML DOM API need to be developed. To target mobile applications, it is required to write symbolic stubs for frameworks used in mobile application implementation.

REFERENCES

[1] C. Cadar, V. Ganesh, P. M. Pawlowski, D. L. Dill, and D. R. Engler, “EXE: Automatically Generating Inputs of Death,” in Proceedings of the 13th ACM Conference on Computer and Communications Security, 2006, pp. 322–335.

[2] C. Cadar, D. Dunbar, and D. Engler, “KLEE: Unassisted and Automatic Generation of High-coverage Tests for Complex Systems Programs,” in Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, 2008, pp. 209–224.

 ISBN: 978-1-61208-367-4

[3] C. S. Păsăreanu and N. Rungta, "Symbolic PathFinder: Symbolic Execution of Java Bytecode," in Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, 2010, pp. 179–180.

[4] K. Sen, D. Marinov, and G. Agha, "CUTE: A Concolic Unit Testing Engine for C," in Proceedings of the 10th European Software Engineering Conference, 2005, pp. 263–272.

[5] N. Tillmann and J. De Halleux, "Pex: White Box Test Generation for .NET," in Proceedings of the 2nd International Conference on Tests and Proofs, ser. TAP'08, 2008, pp. 134–153.

[6] P. Godefroid, M. Y. Levin, and D. Molnar, "SAGE: Whitebox Fuzzing for Security Testing," Queue, 2012, pp. 20:20–20:27.

[7] L. De Moura and N. Bjørner, "Satisfiability Modulo Theories: Introduction and Applications," Commun. ACM, vol. 54, no. 9, 2011, pp. 69–77.

[8] P. Saxena, D. Akhawe, S. Hanna, F. Mao, S. McCamant, and D. Song, "A Symbolic Execution Framework for JavaScript," in Proceedings of the 2010 IEEE Symposium on Security and Privacy, 2010, pp. 513–528.

[9] K. Sen, S. Kalasapur, T. Brutch, and S. Gibbs, "Jalangi: A selective record-replay and dynamic analysis framework for javascript," in Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, 2013, pp. 488–498.

[10] G. Li and I. Ghosh, "PASS: String Solving with Parameterized Array and Interval Automaton," in Proceedings of Haifa Verification Conference, 2013, pp. 15–31.

[11] ECMA International, Standard ECMA-262 - ECMAScript Language Specification, 5th ed., June 2011. [Online]. Available: http://www.ecma-international.org/publications/standards/Ecma-262.htm

[12] "Rhino," https://developer.mozilla.org/en-US/docs/Rhino, [Online; accessed 2014.08.15].

[13] "Use JSDoc," http://usejsdoc.org/index.html, [Online; accessed 2014.08.15].

[14] "JSCover - JavaScript code coverage," http://tntim96.github.io/JSCover/ http://usejsdoc.org/index.html, [Online; accessed 2014.08.15].

 ISBN: 978-1-61208-367-4

Enabling Functional Integration Testing of Software-Intensive Technical Systems by Heterogeneous Models

Thomas Bauer, Frank Elberzhager
Fraunhofer Institute for Experimental Software Engineering IESE
Kaiserslautern, Germany
{thomas.bauer | frank.elberzhager}@iese.fraunhofer.de

Abstract— **In complex software-intensive systems, the analytical quality assurance activities on different levels have become crucial for achieving high product quality. Higher complexity and distributed product development require systematic integration testing to assure interoperability between components and the fulfilment of complex distributed system operations. This work presents the novel automated model-based testing approach ER!S for software-intensive technical systems, which uses a heterogeneous modeling concept for describing the test- and system-specific information. Recommendations from the relevant process standards have been considered to assure and support industrial applicability. The generic approach has been instantiated for functional integration testing on the software design level. It focuses on the functional requirements that are related to distributed system operations implemented by the component interplay. The test model contains the information needed for deriving the test cases for concrete stimulation sequences together with the corresponding expected behavior. The approach supports stepwise system assembly according to an operation-oriented integration strategy. The approach has been initially evaluated in a feasibility study, which was conducted in a research project together with tool vendors and industrial partners from different technical system domains. The first evaluation results are presented. A higher degree of test coverage regarding the relevant functional requirements was achieved.**

Keywords— model-based testing; software integration testing; standard-compliant quality assurance; ISO 26262.

I. INTRODUCTION

The increasing use of software in technical devices like automotive and aerospace systems has enabled the efficient development of new functionality. Software-intensive systems are the main innovation drivers for many embedded system domains nowadays. Most of the innovations are achieved by embedded software [3]. The increasing complexity of software-intensive technical systems regarding their functionality, requirements, system structure, and amount of program code requires more constructive and analytical measures to fulfill the high quality needs within the given economic limits [4].

One consequence of the ever greater complexity of systems and their software parts is the increasing impact of software defects on the overall system quality [2]. A significant number of defects are caused by the faulty interplay of software-controlled components that perform complex functions, the so-called distributed system operations. Therefore, integration and interoperability testing of distributed systems are essential quality assurance activities to check complex sub-system requirements, distributed system operations, and component interaction patterns.

In the research project MBAT, which stands for combined model-based analysis and testing [1], we investigate and develop quality assurance (QA) techniques for safety-related systems from the automotive, avionics, and rail domains.

Development and QA of these systems is guided and driven by different process standards depending on the application domain, e.g., ISO 26262 [6] for passenger cars and DO-178C [7] for airborne systems. Compliance of processes with such standards is an important factor that has to be considered when new technologies are introduced that tackle the challenges of increasing product complexity and economic restrictions.

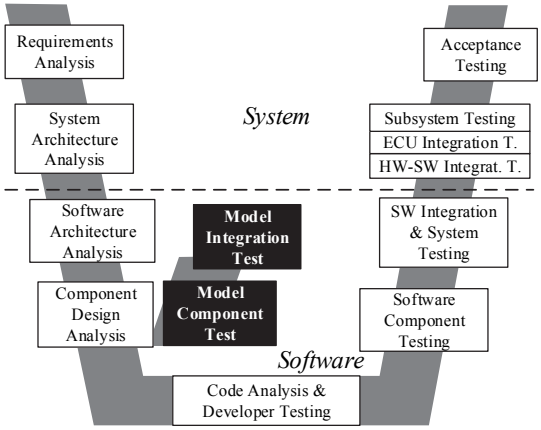


Figure 1. Simplified QA process for software-intensive technical systems

Figure 1 shows a simplified QA process for software-intensive technical systems. Test objects are executable software, software integrated with hardware, and networks of control units driven by software, which leads to various stages of integration testing. Software testing is split into several abstraction levels such as software component, integration, and system testing, where individual components, interacting subsystems, and fully integrated parts are checked against their specifications. If executable models are available from the design stages, dedicated model testing activities on the component and subsystem levels are conducted in addition. This leads to a new branch in the QA process, which is marked in black in the figure.

This article presents the new test approach ER!S for integration testing on the software and model levels of technical software-based systems. It is structured as follows: Section II presents the results from a state-of-the-practice study, which comprises a detailed analysis of the relevant process standards. Section III gives an overview and assessment of the state-of-the-

 ISBN: 978-1-61208-367-4

art approaches in model-based integration testing and motivates why the available solutions are not sufficient. The new model-based testing approach ER!S is presented in Section IV (modeling notation) and Section V (test case generation approach). The article concludes with a short presentation of the evaluation results in Section VI and a summary in Section VII.

II. Analysis of Process Standards

Since technical software-intensive systems perform safety-related functions, standards and guidelines have been developed that are mandatory for product development and QA. Standards provide a high-level overview of the accepted and determined state of the practice at a defined time. They require compliance of the development processes regarding the activities performed and the documents produced depending on criticality degrees. The relevant standards for software development and verification in the transportation domains are IEC 61508 (generic recommendations for safety-related software-intensive systems, [5]), ISO 26262 (for passenger cars, [6]), DO-178C (for avionics systems, [7]), and EN 50128 (for the railway domain, [8]). In our analysis, we focused on ISO 26262 and DO-178C. ISO 26262 re-uses many recommendations and guidelines from the generic standard IEC 61508, which will not be considered separately in this section. The same applies to EN50218, which does not provide additional information for the state-of-the-practice analysis.

Software integration testing is explicitly considered and demanded in ISO 26262 and DO-178C as a mandatory QA activity. ISO 26262 states a set of objectives, coverage criteria, and test case derivation methods. DO-178C and its supplements for model-based development and verification (DO-331) and formal methods (DO-333) define a list of generic objectives to be satisfied during the development and QA of the different artifacts.

Figure 8 in the appendix section shows the relevant recommendations from the two major standards analyzed and the derived and aggregated recommendations focusing on functional software integration testing. The left part of the figure describes the DO-178C recommendations and the right part covers the recommendations from ISO 26262. The generic and aggregated conclusions are described at the bottom of the figure. The corresponding parts, sections, and tables in the documents and the external sources that are referenced in standards are annotated.

Certain recommendations depend on the safety criticality of the artifacts being developed and verified. A higher level of criticality always leads to stricter recommendations and more intensive QA. Both standards have four criticality levels (*A, B, C, D*), but with different orders. The most critical ISO level is D and the highest DO level is A.

ISO 26262 distinguishes between recommended and mandatory actions, which are annotated as lower and upper case letters in the figure. For example, the annotation *abCD* of the ISO recommendation *function coverage* means that this criterion is recommended for criticality levels A and B and mandatory for C and D. The term (*--BA)* for the DO recommendation of the criterion *branch coverage of source code* means that this criterion is recommended for levels A and B, but not needed for levels C and D. Directed arrows show the references between different document parts. Dashed lines represent the relations of the documents and sections of the standards to the generic and aggregated recommendations at the bottom of the figure.

The major recommendation of both standards is the intensive verification and validation of a product's compliance with its requirements. DO-178C, in particular, demands very strict approaches for requirements refinement, traceability, and coverage in the test cases. Additional aspects cover verification of compliance with the software architecture design and the interface definitions, performance properties checks, and coverage of exceptional situations in the robustness test.

Concrete techniques for test case selection are also proposed. Both standards mention the generic, industrially proven, functional testing techniques of equivalence class partitioning, boundary value analysis, and coverage of specification and design models if such model models are available. Detailed criteria and guidelines for the application of these techniques are not provided. An exception is the recommendation of model coverage for finite state machines. The definition of concrete equivalence classes and boundary values and their exploitation for the selection of test cases are not further defined and remain up to the test designer.

Furthermore, architectural coverage regarding component interfaces, interactions, and control and data coupling is recommended. Concrete entity types of interfaces and interaction elements are not mentioned. From the perspective of complex system operations, the coverage of functions, function calls, and function sequences is demanded. This criterion is important for the coverage of complex use cases and distributed system operations.

ISO 26262 and DO-178C explicitly support the use of formal models, especially behavior models, for development and QA activities. Different notations are mentioned and recommended, for example transition-based notations (finite state machines), pre-/post-based notations (Z), and operational and concurrent notations (Petri nets). More information on the modeling notations and their classification is provided in [9].

The conclusions for functional integration testing on the model and software levels are: Requirements coverage is the major criterion and the most important goal of testing. Concrete techniques and criteria for creating requirements-based test cases are mentioned. The specification of complex and distributed system operations represents high-level requirements of the integrated system that have to be checked intensively. Additionally, the coverage of the software architecture as well as that of component interfaces and their interactions has to be considered for test design and test specification. Based on our experience from the MBAT project, no industrially proven automated test approach is available that sufficiently and efficiently covers standard-compliant functional integration testing on the model and software levels.

III. State of the Art

Model-based approaches provide a high degree of automation regarding analysis, transformation, and generation of valid execution sequences due to their sound mathematical basis. Different modeling notations have been systematically exploited for the generation of test cases. The set of corresponding techniques is called *model-based testing* (MBT). MBT approaches address those 40% of testing effort that are usually

 ISBN: 978-1-61208-367-4

spent on test preparation and test specification in an industrial project [32]. By automating these activities, the overall testing effort can be significantly reduced. The standards analyzed in the previous section also propose the use of models for development and QA activities. The main focus of the MBT techniques is on functional testing, i.e., on testing against the functional specification [9]. Functional testing usually requires the specification of test cases with the corresponding pre-conditions, actions, expected results, and post-conditions.

Models that are constructed for the primary use of generating test cases from them are called *test models.* They represent the relevant information from the test and QA perspectives. Test models can describe intended and unintended functionality, unexpected and unspecified usage, or misuse to support robustness testing. Additionally, test models also guide and facilitate test case generation by providing information on importance and criticality or on the frequency of certain scenarios. An overview of test modeling notations and test case generation approaches is provided by Utting et al. in [9]. For model-based integration testing, numerous approaches have been developed with different kinds of test objectives and modeling notations. A detailed overview and a classification are given by Bauer and Eschbach in [31]. The approaches have been classified intro three classes (component-based, scenario-based, and combined approaches), which are described below. There are additional approaches that work directly on program code and code-based integration testing, but they are not considered here due to restricted code access in most industrial projects and missing support for testing high-level requirements.

The remaining solutions of the three classes have been assessed regarding their capabilities for modeling the two dimensions of integration testing: the low-level *interaction-focused view* and the *high-level requirements view,* which is related to *distributed system operations*. Due to the high complexity of the software systems and their requirements, a *stepwise assembly strategy* and the *composition of operations* should be supported. Test scenarios require the specification of their pre- and post-conditions. Therefore, the notation should also support the modeling of such *execution conditions*. Finally, the approach should be able to describe the *interaction patterns* regarding the system components and their interfaces as part of the operational implementation.

The *component-based integration test approaches* use dedicated behavior models, mostly different types of finite state machines, from the component perspective as the basis for test case generation. Their origin is the conformance and interoperability testing of protocols [10]. For integration testing, different finite state machine notations are used to represent the system behavior. Most of them focus on the coverage of synchronized events, e.g., the approaches by Koppol et al. [12] and Robinson-Mallett et al. [14]. Other approaches use extended finite state machines with variables and guards and define specific criteria for additionally covering data coupling and data flow dependencies on the subsystem level [13][15][16]. Component-based approaches usually support stepwise system assembly and integration testing. The main problem is that complex scenarios and high-level requirements are not sufficiently considered due to the focus on specific component interactions.

The *scenario-based approaches* focus on the modeling of high-level system requirements, system operations, use cases, and usage scenarios. Most of them use UML behavior diagrams such as sequence diagrams, collaboration diagrams, interaction or activity diagrams [18][19]. Each scenario (including rare and exceptional cases) has to be modeled explicitly. A second group applies operational modeling notations that consider concurrency like Petri nets [20] and Communicating Sequential Processes (CSP) [21]. The operational modeling notations have advantages in terms of model composition, but weaknesses regarding the description of operational execution conditions. Due to the high-level view focusing on usage scenarios, low-level aspects such as concrete component interfaces and component interactions are not covered sufficiently by all scenario-based approaches. The strategy of stepwise assembly is not considered by any of the approaches.

The most advanced solutions consider the heterogeneous aspects of functional integration testing: the high-level system features, operations, and requirements on the one hand and the concrete component interactions on the other hand. The approaches are classified as *combined integration test approaches*. They use different kinds of finite state machines to model the low-level behavior on the component and subsystem levels and a high-level model to describe the relevant usage scenarios and high-level requirements. For modeling the scenarios, different notations are used, such as finite state machines in the approach by Wieczorek et al. [24], UML collaboration diagrams in the solution by Ali et al. [22], or a tree-like feature interaction model in the publication by Benz [23]. All approaches support at least simple solutions for the composition, the description of the operational execution conditions, and the modeling of component interactions as operational implementations, but no approach exists that completely covers all aspects to the full extent. However, the approach by Benz [23] supports different kinds of composition operators and the one by Wieczorek et al. [24] supports the detailed modeling of component interactions.

The conclusion of the state-of-the-art analysis is that heterogeneous integration test approaches provide the most appropriate solutions for our problem. They are able to cover high-level system operations as well as low-level component interactions. None of the state-of-the-art approaches sufficiently supports all requirements stated. The composition of system operations, the modeling of complex execution conditions, and their implementation as component interplay is only partially solved by the available solutions. Therefore, we have developed a new model-based test approach that tackles these challenges.

IV. TEST MODELING NOTATION

The efficiency of MBT highly relies on the selection of an appropriate modeling notation and the availability of efficient model analysis technologies. The notation influences the quality and efficiency of model construction, i.e., the formalization of the requirements, and test case generation, i.e., the derivation of traces from the model.

The selection of an appropriate modeling notation depends on the characteristics of the system and its functionality to be modeled. As described above, the application type is the software level of embedded systems. In embedded systems, two types of functions are usually provided: computation and control functions. Computation functions are mainly used for connecting the system with its environment via sensors and actuators and for deriving relevant variables and decision points.

 ISBN: 978-1-61208-367-4

Control functions are connected to the system state and modes. They are usually used at a higher abstraction level than computation. Based on the stimulation pattern and the current system state, the future behavior is controlled.

MBT often focuses on testing the functional behavior and the control functionality. The functional behavior is expressed by stimuli, responses, pre- and post-conditions, and state variables. Especially for technical software systems, inputs and outputs can be complex due to time dependency and concurrency. Solutions have been proposed particularly for the Simulink/Stateflow simulation environment [26]. The following subsections will describe the generic heterogeneous test modeling approach (in subsection A), its instantiated modeling notations (in subsections B and C), and the concept for assuring consistency of the two notations (in subsection D).

A. Towards a generic test modeling approach

In order to enable fully automated test case generation, test models have to describe system-specific aspects, such as the system structure and component interfaces, as well as test-specific aspects, such as the importance of scenarios, interfaces, and modes. Most MBT approaches use one modeling notation as a basis for generating test cases [9]. In order to clearly divide the responsibilities of the model artifacts, the *generic heterogeneous test modeling approach* ER!S has been developed. The approach distinguishes between a low-level model that represents the test-relevant system behavior and a high-level model for representing complex requirements and guiding test case derivation. In the MBAT project, the approach has been instantiated for functional software integration testing.

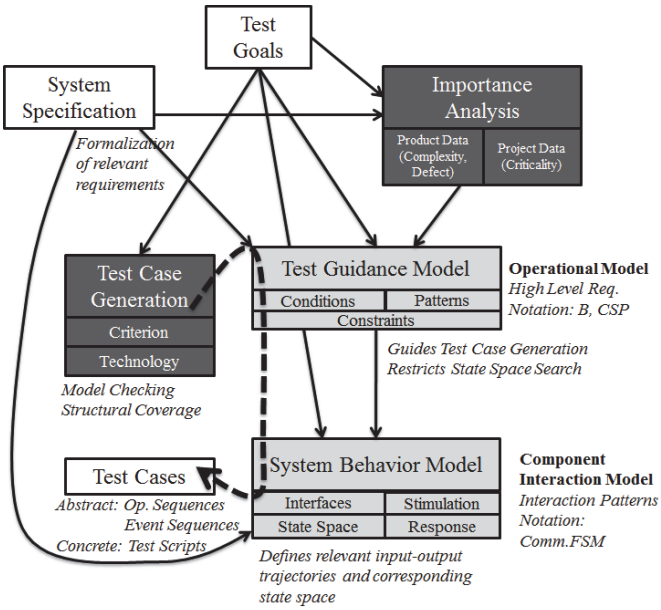


Figure 2. Generic test modeling approach

Figure 2 describes the artifacts involved in the generic test modeling approach and their relations. The starting point is the *system specification*, which is the initial source for describing the system functionality to be checked. In most cases, the system specification is a textual requirements document enriched with architectural descriptions of the components, interfaces, and communication middleware. The *test goals* describe the generic objectives of quality assurance, such as coverage of the functional requirements, assuring robustness in unspecified situations, or considering the most critical usage scenarios. Test goals influence the QA strategy and therefore also test modeling and test case generation in MBT.

Based on the specification and the test goals, an *importance analysis* is conducted, which considers the complexity and defect data of the product and other criticality factors of the test project. The importance analysis influences the abstraction level of the test modeling and the inclusion and exclusion of elements and requirements. Additionally, the importance analysis may serve as input for the distribution of the test effort, the selection of the test coverage criteria, and the guidance regarding test case generation.

In the approach, two types of test models are constructed: the *system behavior model* (SBM) and the *test guidance model* (TGM). The SBM defines the relevant system behavior for the test on an appropriate abstraction level. The SBM is constructed from the system specification and describes the interfaces, valid input-output trajectories, and the state space of the test object. For this work, the SBM focuses on component interactions. Therefore, the SBM for functional integration testing is also called *interaction test model* (ITM). Due to the characteristics of the actual test object of the evaluation, a discrete control system, the modeling notation for the ITM is a subset of timed automata [29]. This notation enables the description of a component-based system whose parts are synchronized by events. Communication protocols and specific middleware entities such as bus controllers can be modeled as additional state-based components of the SBM.

The TGM describes patterns and constraints for the application and exploration of the SBM and the conditions under which they are to be applied. Constraints are defined to prohibit or enforce defined situations for the forthcoming test case generation. The TGM can also represent the operational profile of the test object, which may differ in different environments. For this work, the TGM has to deal with composite system operations and functional scenarios with defined execution conditions. Therefore, the TGM for functional integration testing is called *operational test model* (OPM). A concrete operational implementation is defined by the interaction patterns of the system components. Due to the strong focus on the composition of different operations and the efficient description of their execution conditions, a heterogeneous notation based on B machines [27] and CSP [28] has been chosen. The integration of B machines and CSP for formal verification purposes has been shown in [30].

For the *test case generation* step, the coverage criteria and the generation technology have to be defined [9]. Considering the ER!S models, the coverage criteria determine the class of relevant elements of the test model that shall be covered by the test cases. Examples are the coverage of component interfaces or the coverage of conditional execution paths within an operation. The test cases are represented by sequences of operations and events, which are refined to executable test scripts (like C-Unit scripts [38] or signal descriptions in the Matlab / Simulink environment [37]).

B. The Operational Test Model

The OPM is used to guide the selection of test cases from an operational point of view. It describes the high-level functional requirements and system operations with their composite

 ISBN: 978-1-61208-367-4

implementations and execution conditions, and the system state space. The implementation of the modeling notation is based on B machines and CSP. The OPM is defined as: $OPM = \{Op, Var_{Op}, S_{OP}, s_0, S_{Exit}\}$. It contains a finite set of hierarchical operations *Op*. The composition relation is defined by a partial ordering function, which enables operational composition with different operators. These operators are based on the composition operators of CSP [28]: sequencing, alternatives, conditional branching, and parallel interleaving. They enable the construction of so-called *composite operations*. The non-composite operations are called *basic operations*. A finite set of variables Var_{OP} is defined to model the system states and operational execution conditions. Combinations of variable values define the system state space. A dedicated start state s_0 and an optional set of ending states S_{EXIT} for the execution of test case are defined. For the OPM, two graphical representations have been developed to facilitate discussions and model reviews: the Operational Hierarchy Graph (OHG) and the Operational Composition Graph (OCG).

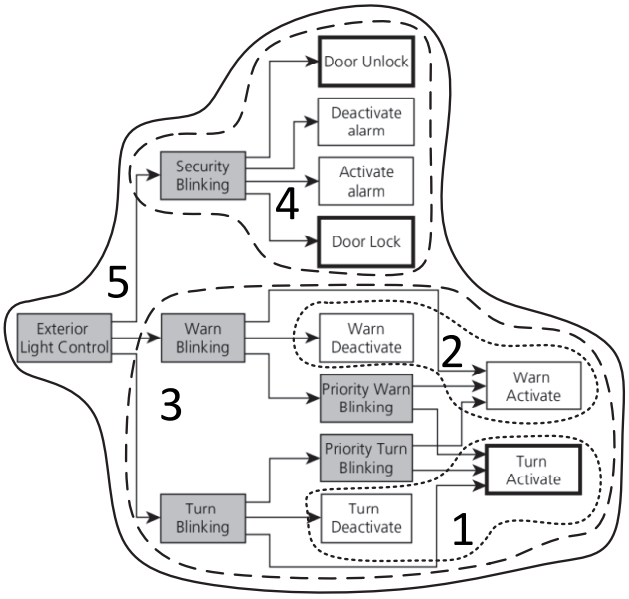


Figure 3. Operational Hierarchy Graph for the sample application

Figure 3 shows an OHG, which represents the hierarchy of operations regarding their composition relations. Boxes represent operations. Basic operations are marked white, composite operations are marked gray. The arrow points to the sub-operations of a composite operation. The example shown is taken from the evaluation case study described in chapter VI. Additionally, a valid integration order for the system operations is annotated, which consists of five steps. The integration is performed from lower-level to higher-level operations, i.e., from step #1 to step #5.

The OCG visualizes the operational composition with a directed graph. An example of an OCG is shown in Figure 4. It describes the composite operation *Warn Priority Blinking* of the example used in the feasibility example. The rectangular boxes represent the sub-operations referenced and rounded rectangles represent the execution conditions (here: pre- and post-condition) with the corresponding Boolean formulas. Composition operators are shown with specific symbols, such as arrows for sequencing and diamonds for alternatives. The OCG traversing starts in the pre-condition node and ends in one of the post-condition nodes. Every trace through an OCG is a valid operational execution.

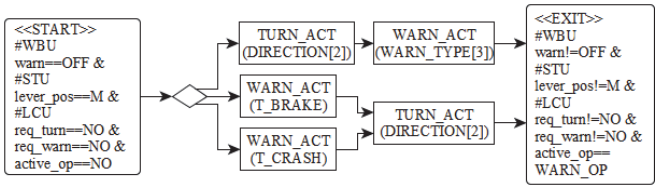


Figure 4. Operational Composition Graph of an operation

The operation of the example deals with the determination of the active blink operations when multiple turn and warn blink operations (manual, emergency brake, and crash) are requested. The interesting cases in the example are when (1) a previously activated turn blink operation is overwritten by a subsequent warn blinking (manual, emergency brake, and crash) and (2) a previously activated emergency brake and crash warn blinking is deactivated by subsequent turn blinking.

C. The Interaction Test Model

The ITM describes concrete interactions between system components in order to implement an operation. Its notation is a subset of timed automata [29]. The ITM is defined as the parallel composition of a set of component models (*CM*) that may synchronize on shared events. A CM is defined as $CM = (L, l_0, Var_{Comp}, Act, E$. Locations (*L*) represent the vertices of the component automaton connected by a set of edges *(E)*. Every CM has a designated initial location (l_0) and a set of variables (Var_{comp}), which is a subset of the operational system variables (*Var*). Furthermore, a CM has an alphabet of events with inputs and outputs (*Act*). Edges (E) connect two locations. They are annotated with the corresponding input (?) or output (!) event. The operations are implemented by a set of component interactions, which are related to concrete component edges. Therefore, the ITM component edges are annotated with the set of operations that are connected to them.

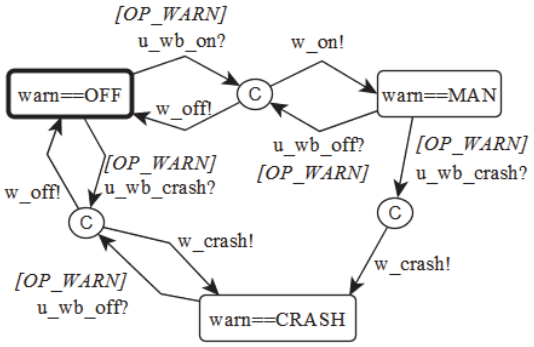


Figure 5. Sample interaction test model of a component

Figure 5 shows a sample ITM for the component *WarnBlinkUnit* of the feasibility study. The graphical representation is similar to common finite state machine notations. The ITM locations are expressed as rounded nodes

 ISBN: 978-1-61208-367-4

(stable states) or circles (committed states); the transitions are represented as directed edges. The stable component states are annotated with an invariant, which is a unique combination of component state variable values. In the example, only one variable is used. A state is stable if all of its outward transitions are only enabled by external stimulation [33].

Transitions are labeled with at most one event, which is either a sending event (!) or a receiving event (?). The relevance of transitions for the implementation of certain operations is annotated by guard conditions. In our example, the model transitions are used for the warn blink operations, which are expressed by the Boolean variable *OP_WARN*. For analysis and test case generation, the transition guards help to reduce the complexity of the artifacts and focus on the relevant parts.

The component behavior is defined by a sequence of one input and a list of outputs, which have stable source and target states. Since timed automata allow only one event per transition, the input and the corresponding output(s) are constructed as an atomic sequence of transitions connected by committed states. During the exploration of the system state space, committed states have to be left immediately by taking an outgoing transition when they are traversed. This assures the atomicity of the event sequence.

D. Model Construction and Analysis

Complex formal models that are created manually from potentially incomplete and inconsistent sources require a systematic construction process and intensive QA. Another issue is the use of different model types, which may produce consistency issues. The construction approach provides a systematic procedure for designing the test model and applies guidelines and restrictions to reduce the fault-proneness and complexity of the artifacts. A formal correctness proof of a complex model is difficult to achieve. Therefore, a stepwise heuristic procedure is applied, comprising parallel construction and model analysis activities. The approach is supported by prototypical tools. For the conduction of the model analysis, the external tools Uppaal [35] and ProB [36] were used.

Both models, OPM and ITM, were checked independently regarding certain properties such as deadlocks and reachability of elements. Further analysis activities assured the consistency of both models. A catalogue of concrete analysis activities was defined, which is described in part below.

As shown in Figure 2, the main source for the test modeling is the system specification, which contains all information about the static system structure of the test object and its functionality and operations. The construction approach of ER!S models was derived from sequence-based specification (SBS, [34]) which enables the systematic specification of component test models. The system functionality in ER!S is specified as a set of operations that are implemented as interactions between components under defined conditions.

The recommended construction approach from the operational view is bottom-up. According to the operational hierarchy, basic operations are specified first with their execution conditions and interaction patterns. These interaction patterns describe event flows, sequences of inputs and corresponding outputs, and conditions under which they are applicable. The system is supposed to run in a so-called slow environment [33]. This means that the system is only stimulated when all of its components are in stable states, i.e., the components do not perform autonomous interactions. All component responses are direct reactions to stimulation from the environment. Operations always start and end in stable system states, which facilitates the construction of deterministic test models. This leads to special requirements for the event sequences and states that are checked in the ITM analysis. Furthermore, the ITM is checked for interoperability, i.e., the ability of communicating via its interfaces.

In the next step, the composite operations with their composition patterns and the execution conditions are specified. In the subsequent analysis, the OPM is checked for the validity and executability of the composition patterns. The OPM analysis checks whether operational traces exist that completely traverse the operational specification.

The quality of the source documents affects the construction paradigm of the ER!S models. Faulty, inconsistent, incomplete, or even changing requirements lead to model design flaws and model changes. In order to assure compatibility and consistency between different modeling notations, two concepts are introduced that focus on the relations between operations and interaction. The first concept is an *injective mapping function* for OPM states and ITM states. Each state of the OPM state space is mapped to a unique stable system state of the ITM. The reachability of selected stable states of the ITM is checked. Specific requirements for stable states regarding variable values and transition events are defined and checked as well. The second concept are *operational tags*, which are annotated to ITM component transitions. For each operation, the corresponding sub-model of the ITM is determined. The ITM analysis assures that the interaction patterns of the operations are executable and valid regarding the conditions and variable values.

V. GENERATION OF TEST ARTIFACTS

After the construction of the test model and its verification, test cases are derived as ER!S model traces. The test case set comprises valid model traces that cover selected test requirements. For the generation of test cases, many results of the ER!S model analysis can be reused since they contain sample model traces that prove the reachability of defined model elements.

Figure 6 shows the generation of test artifacts from the test models. The starting point are the TGM and the SBM, which are analyzed in order to define the *assembly strategy* for the system components and operations. An operation-driven bottom-up strategy is proposed, i.e., operations of a lower hierarchy level are integrated earlier, whereas complex operation of higher levels are integrated later in the test stage [31]. An example of a valid assembly strategy for a feasibility study is annotated in Figure 3. Each step covers a disjoint set of the selected test requirements. Two criteria have been developed that enable the scalable assembly of components and operations. The first criterion determines the relevant sub-systems that perform specific operations. The other criterion determines the integration strategy for the relevant system

 ISBN: 978-1-61208-367-4

operations in each integration step. Since each assembly step focuses only on specific aspects of the system functionality, only a subset of the test models is required for test case generation. Therefore, step-specific reduced test models are created that only contain the required subset of the information.

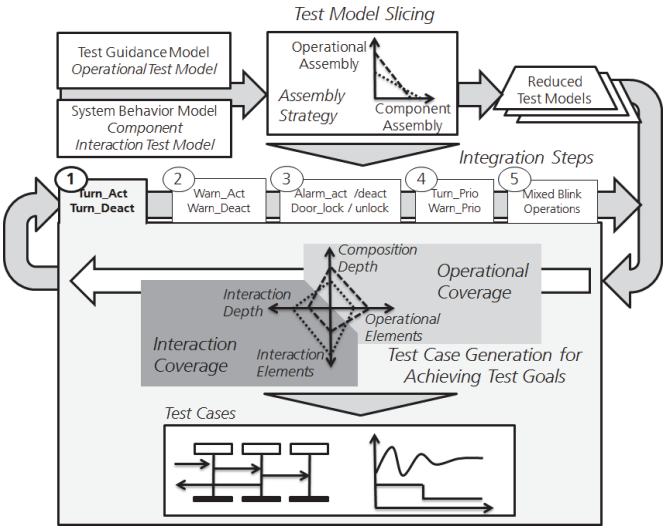


Figure 6. Workflow for generating test artifacts

System operations have different kinds of test-relevant information depending on their implementation and composition. Basic operations focus on the coverage of interaction patterns under specific conditions. Composite operations focus on the coverage of their alternative composition paths under specific conditions. Therefore, our test approach provides a set of *coverage criteria* for component interactions and for operational composition. The interaction coverage criteria are related to component interfaces, event-sending and receiving transitions, and synchronizations of them. The operational coverage criteria consider the different execution conditions and the characteristics of the composition operators used in the implementation.

The test case generation procedures use the model checking capabilities of the tools Uppaal and ProB, which are capable of verifying properties and deriving sample traces for timed automata, B machines, and CSP models. The ER!S tool transforms the coverage criteria and sets of test requirements into simple model checking queries for the tools mentioned above. The resulting test cases are valid ER!S model traces. They consist of sequences of operations implemented by event traces. More details on the test selection criteria and generation technologies are provided by Bauer and Eschbach in [31].

VI. EVALUATION

The evaluation was conducted in the MBAT project together with tool vendors and product manufacturers from the transportation domains. The goal was to assess the impact of the new heterogeneous MBT technique on the test process compared to manual expert-driven requirements-based test case creation and a simple MBT technique with finite state machines, which had been already introduced to the companies. The properties to be evaluated were: (A) compliance with the recommendations of the process standards, (B) coverage of the test cases regarding the properties to be checked, and (C) the manual effort spent on the construction of the test artifacts.

The evaluation was planned to be conducted in two rounds: (1) a feasibility study to initially assess the new test approach and (2) a detailed quantitative evaluation study to measure the impact. In the following subsection, the test object, the results of the first evaluation round, and the set-up for the second round are presented.

There were several challenges that complicated an evaluation in the MBAT project. The first challenge was the missing independence of system experts and test experts (for the expert-driven requirements-based test case derivation), which might have influenced the significance of the results. The other issue was the confidentiality of the test object in the project, which restricted the usage and publication of certain details. Therefore, a new test object with the corresponding specification documents, design models, and executables was created. The functionality is close to the features of the actual test object, but the system structure, component interfaces, and the concrete implementations were simplified and developed independently to abstract from any confidential details.

A. The test object

The test object of the evaluation case study was a simplified version of an executable design model (notation: Simulink / Stateflow) of an automotive exterior light control system (ELCS). The ELCS consists of five system components: steering unit, ignition unit, warn blink unit, door control unit, and exterior light control unit. Another eight environmental components were considered for the evaluation of stimulation and response. The functionality comprises several blinking functionalities of the external lighting, including turn indication, warn blinking, and security features such as door locking and theft alarm.

TABLE I. PRIORITIES OF BLINK OPERATIONS IN THE CASE STUDY

Prio	Class	Blink operation	Duration	Side
1	Warning	Crash warning	Permanent	Both
2		Manual warning*	Permanent	Both
3		Brake warning	Permanent	Both
4	Turn indication	Permanent*	Permanent	Left/right
5		Temporary*	Temporary	Left/right
6	Security	Theft alarm	Permanent	Both
7		Door locking	Temporary	Both
8		Door unlocking	Temporary	Both

Table I shows the different blink functionalities with their priorities (1 – highest, 8 – lowest) and the properties that were checked in the test evaluation. Several blink operations can be requested at the same time, but only one operation can be active. If multiple blink operations are requested, the operation with the highest priority is selected and executed. The only priority exception in the example application is that turn indication overwrites an active warn blinking in certain situations (marked with * in the table).

Each functionality has defined pre-conditions for its activation and deactivation, for example regarding the ignition status or door locking status. The observable outputs of each functionality are the flashing side markers outside the car and

 ISBN: 978-1-61208-367-4

the flashing LEDs on the car's dashboard at defined frequencies. The key issue of functional integration testing is to assure the correct execution of the blinking behavior in the case of multiple activated blinking functions.

B. Preparation of the evaluation

As the first step of the evaluation, a simplified requirements specification of the ELCS was developed based on the knowledge of system and test experts, the original requirements, and the existing test goals and test cases. The new specification does not contain confidential information and abstracts from irrelevant details for functional integration testing. Based on the simplified specification, the executable test object, i.e., the application under test, was developed as an executable Matlab / Simulink model [37], which enables the automated generation of program code. In the specification and design activities, experts with system and domain knowledge were involved as well as test experts.

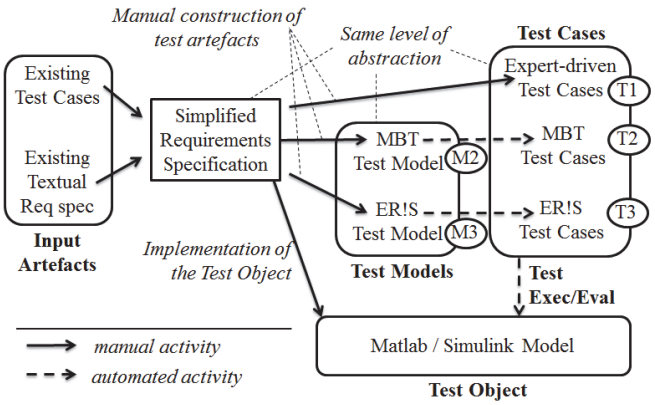


Figure 7. Evaluation set-up

Figure 7 shows the set-up of the evaluation, which comprises the development of the actual test object, the construction of the test artifacts, and the derivation of the test cases for the following three methods: (1) manual, expert-driven, and requirements-based test case derivation, (2) MBT with finite state machines, and (3) MBT with heterogeneous test models (the ER!S approach).

After the creation of the test object, the test artifacts for the different approaches were created based on the same simplified requirements specification. The manual creation of the expert-driven test cases (T1) was done by system experts according a standard-compliant, expert-driven, and requirements-based approach. The construction of the test models for the MBT with finite state machines (M2) and the MBT with heterogeneous models (M3) was done by dedicated method experts. For the construction of the test artifacts (models and test cases), the same abstraction level regarding system structure, component interfaces, events, and variables was applied. The test cases for both MBT approaches were created automatically, which is displayed as dashed lines in the figure.

C. Results from the feasibility study

In the first evaluation round, we aimed at a short assessment of the test technology. Therefore, we considered a subset of the systems' functionality. Test models and test cases were created by applying each of the three methods. The assessment regarding the selected properties is summarized in Table II. The complexity of the different test artifacts could not be assessed adequately since the test approaches use different modeling paradigms (test sequences, finite state machines, and the heterogeneous notation based on timed automata, B, and CSP).

TABLE II. SUMMARY OF FIRST EVALUATION RESULTS

	Expert/req-based (T1)	**MBT-FSM (M2, T2)**	**ER!S (M3, T3)**
Standard Compl.	+	-	+
Operational Test Coverage	81%	67%	90%
Interaction Test Coverage	92%	100%	100%
Test Effort	100%	119%	135%

The first evaluation aspect is standard compliance, which qualitatively assesses the considerations of the recommendations from the process standards mentioned in section II. The ER!S approach was developed with the intent of being compliant with the industrial standards and guidelines. The support of certain topics, such as coverage of functional requirements, interactions, and operational sequences, is sufficient and comparable to the expert-driven test approach that has been applied to the test project and the resulting certification for many years. The MBT approach with finite state machines does not sufficiently consider the characteristics of more complex system operations.

The next aspect is test coverage, which is a quantitative quality criterion of a set of test cases regarding a set of properties and test requirements. It can be seen as an indicator of the quality of the test process. The ER!S approach facilitates the determination of appropriate criteria regarding the component interactions and the operational implementation. The initial evaluation of interaction and operational coverage showed that ER!S test cases (T3) achieved high coverage of both criteria (100% regarding the interactions and 90% regarding the operational aspects). The MBT test cases (T2) achieved full coverage of the selected interactions, which is explained by the strong focus on component behavior and communication. Therefore, the operational coverage of T2 is also much lower (67%) than with the other approaches. The expert-based test cases (T1) had reasonably high coverage of both criteria (92% regarding the interactions and 81% regarding the operational aspects). An influencing factor for the detailed assessment is the varying degree of importance of selected test requirements, which was not considered in the initial evaluation. The discussion with the industrial partners showed that the automated test approaches with their test case sets T2 and T3 contained a slightly higher number of less relevant elements. In the next evaluation round, a more detailed analysis of the test coverage will be conducted.

The reduction of effort and costs is an important success factor when it comes to introducing new technologies. In our feasibility study, the effort for manually constructing the test artifacts was assessed (T1, M2, M3). The main part of the effort for all approaches was spent on determining and defining the components, interfaces, events, variables, and interaction patterns in order to ensure the same origin and abstraction level

 ISBN: 978-1-61208-367-4

of the test cases. Automated steps, such as the test case generation for T2 and T3, were not considered. The initial effort for constructing the first set of test models and test cases (+19% for M2 and +35% for M3) is slightly higher for the model-based approaches than for the traditional testing approach (T1). This is caused by the fact that T1 only contains selected scenarios for the application. Test models and the resulting test case sets are often more complete regarding the selected properties. A significant effort reduction for the model-based approaches is expected when existing test artifacts are incrementally extended for updated product versions and similar systems.

The limitations of the feasibility study only enable an initial and rough assessment of the impact and capabilities of our ER!S approach. The standard compliance, the higher test coverage, and the slightly higher test effort in the first round are indications that ER!S is an efficient and reasonable integration testing approach. Further evaluations are needed to assess the capabilities and impact on the overall test processes in detail.

VII. Conclusion and Future Work

In this article, we presented the novel model-based approach ER!S for functional integration testing of software-intensive technical systems. The detailed analysis of the recommendations and challenges stated in the two relevant process standards (ISO 26262 and DO-178C) resulted in a set of major requirements for functional software integration testing that can be addressed by model-based solutions. The state-of-the-art approaches do not sufficiently cover the multifaceted aspects of integration testing with the two dimensions of composite and distributed system operations and the actual component interplay that implements these operations.

The ER!S approach is able to efficiently tackle these challenges. It comprises a heterogeneous modeling notation that considers both aspects of functional integration testing. The notation enables the automated generation of test cases using different coverage criteria. The results of the first evaluation round were positive. Our approach produced test artifacts of higher test quality regarding the test coverage. More detailed results will be gathered in the second evaluation round. Other future activities will comprise the improvement of the analysis and test case generation algorithms and the extension of the tool chain, which currently consists of a set of loosely coupled in-house, external research, and commercial tools.

Acknowledgment

This work has been funded by the ARTEMIS project “MBAT” (grant number: 269335). We would also like to thank Sonnhild Namingha for proofreading.

References

[1] Website of the project MBAT,,http//www.mbat-artemis.eu [12 Aug 2014]

[2] P. Liggesmeyer and M. Trapp, “Trends in Embedded Software Engineering”, IEEE Software 26(3), pp. 19-25, Jan. 2009.

[3] M. Maurer and H. Winner, Automotive Systems Engineering, Springer June 2013

[4] J. Zander, I. Schieferdecker, P. Mosterman, Model-Based Testing For Embedded Systems, CRC Press, Sep. 2011

[5] IEC 61508, International Electrotechnical Commission, IEC 61508:2010 - Functional safety of electrical/electronic/programmable electronic safety related systems, 2010

[6] ISO 26262, International Standardization Organization, ISO 26262:2011 - Road vehicles – Functional safety, 2011

[7] DO-178C,.Radio Technical Commission for Aeronautics Software, DO-178C:2011 Considerations in Airborne Systems and Equipment Certification, 2011

[8] EN 50128, CENELEC - European Committee for Electrotechnical Standardization, EN 50128:2011, Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems, 2011

[9] M. Utting, A. Pretschner, B. Legeard, “A Taxonomy of Model-based Testing Approaches”, Softw. Test. Verif. Reliab. 22(5), pp. 297-312, Aug. 2012

[10] A. V. Aho, A. T. Dahbura, D. Lee, M. Uyar, “An optimization technique for protocol conformance test generation based on UIO sequences and rural Chinese postman tours”, IEEE Transactions on Communications 39(11), 1604-1615, Nov. 1991

[11] Y. Wu, D. Pan, M.-H. Chen, “Techniques for Testing Component-Based Software”, Proceedings of the 7th Int. Conf. on Engineering of Complex Computer Systems, IEEE Computer Society, pp. 222-232, June 2001

[12] P. V. Koppol, R. H. Carver, K.-C. Tai, “Incremental Integration Testing of Concurrent Programs”, IEEE Trans. Software Eng. 28(6), pp. 607-623, June 2002

[13] A. Desmoulin and C. Viho, “A New Method for Interoperability Test Generation”, Proceeding of the TestCom/FATES'07, Springer, pp. 58-73, June 2007

[14] C. Robinson-Mallett, R. Hierons, J. Poore, P. Liggesmeyer, “Using Communication Coverage Criteria and Partial Model Generation to Assist Software Integration Testing”, Software Quality Control 16(2), pp. 185-211, Apr. 2008.

[15] L. Gallagher, J. Offutt, A. Cincotta, “Integration testing of object-oriented components using finite state machines”, Softw. Test., Verif. Reliab. 16(4), pp. 215-266, Jan. 2006

[16] F. Saglietti, N. Oster, F. Pinte, “Interface Coverage Criteria Supporting Model-Based Integration Testing”, ARCS '07 - Workshop Proceedings, VDE Verlag GmbH , pp. 85—93, 2007

[17] A. Pretschner, O. Slotosch, E. Aiglstorfer, S. Kriebel, “Model-based Testing for Real: The Inhouse Card Case Study”, Int. J. Softw. Tools Technol. Transf. 5(2), 140—157, 2004.

[18] A. Abdurazik and J. Offutt, “Using UML Collaboration Diagrams for Static Checking and Test Generation”, Proceedings of the 3rd International Conference on The Unified Modeling Language: Advancing the Standard, Springer-Verlag, pp. 383—395, 2000

[19] F. Basanieri and A. Bertolino, “A Practical Approach to UML-based Derivation of Integration Tests”, in 4th International Software Quality Week Europe, Nov. 2000

[20] H. Reza and E. Grant, “A Method to Test Concurrent Systems Using Architectural Specification”, J. Supercomputing 39(3), pp. 347-357, Feb. 2007

[21] S. Nogueira, A. Sampaio, A. Mota, “Guided Test Generation from CSP Models”, Proceedings of the 5th International Colloquium on Theoretical Aspects of Computing, Springer, pp. 258-273, Sep. 2008

[22] S. Ali et al., “A State-based Approach to Integration Testing Based on UML Models”, Inf. Softw. Technol. 49(11-12), pp. 1087-1106, Nov. 2007

[23] S. Benz, “Combining Test Case Generation for Component and Integration Testing”, Proceedings of the 3rd International Workshop on Advances in Model-based Testing, ACM, pp. 23—33, May 2007

[24] S. Wieczorek et al., “Applying Model Checking to Generate Model-Based Integration Tests from Choreography Models”, 'Testing of Software and Communication Systems', Springer, pp. 179-194, Nov. 2009

[25] M. Utting, A. Pretschner, B. Legeard, “A Taxonomy of Model-based Testing Approaches”, Softw. Test. Verif. Reliab. 22(5), pp. 297-312, Aug. 2012

[26] F. Böhr, Model-based Statistical Testing of Embedded Real-time Software with Continuous and Discrete Signals in a Concurrent Environment: The Usage Net Approach, PhD thesis, TU Kaiserslautern, Dr. Hut, 2012

 ISBN: 978-1-61208-367-4

[27] J.-R. Abrial, The B-book: Assigning Programs to Meanings, Cambridge University Press, 1996

[28] C. A. R. Hoare, Communicating Sequential Processes, Prentice-Hall, Apr. 1985

[29] R. Alur, and D. L. Dill, "A Theory of Timed Automata", Theoretical Computer Science 126 (2), pp. 183-235, Apr. 1994.

[30] M. J. Butler and M. Leuschel, "Combining CSP and B for Specification and Property Verification", Proceedings of the 2005 international conference on Formal Methods, Springer, pp. 221-236, July 2005.

[31] T. Bauer and R. Eschbach, "Model-Based Testing of Distributed Functions", Advanced Automated Software Testing: Frameworks for Refined Practice, CRC Press, pp. 151-181, Jan. 2012

[32] M. Pol, Software Testing: A guide to the TMap Approach, Addison-Wesley Professional, Nov. 2001

[33] M. Shahbaz, Reverse Engineering Enhanced State Models of Black Box Components to support Integration Testing, PhD thesis, Grenoble Institute of Technology, 2008

[34] S. J. Prowell, and J. H. Poore, "Foundations of Sequence-Based Software Specification", IEEE Trans. Software Eng. 29(5), pp. 417-429, May 2003

[35] Website of Uppaal, http://www.uppaal.org/ [12 Aug 2014]

[36] Website of ProB, http://www.stups.uni-duesseldorf.de/ProB [12 Aug 2014]

[37] Website of Simulink http://www.mathworks.de//simulink [12 Aug 2014]

[38] Website of C-Unit http://cunit.sourceforge.net/ [12 Aug 2014]

APPENDIX

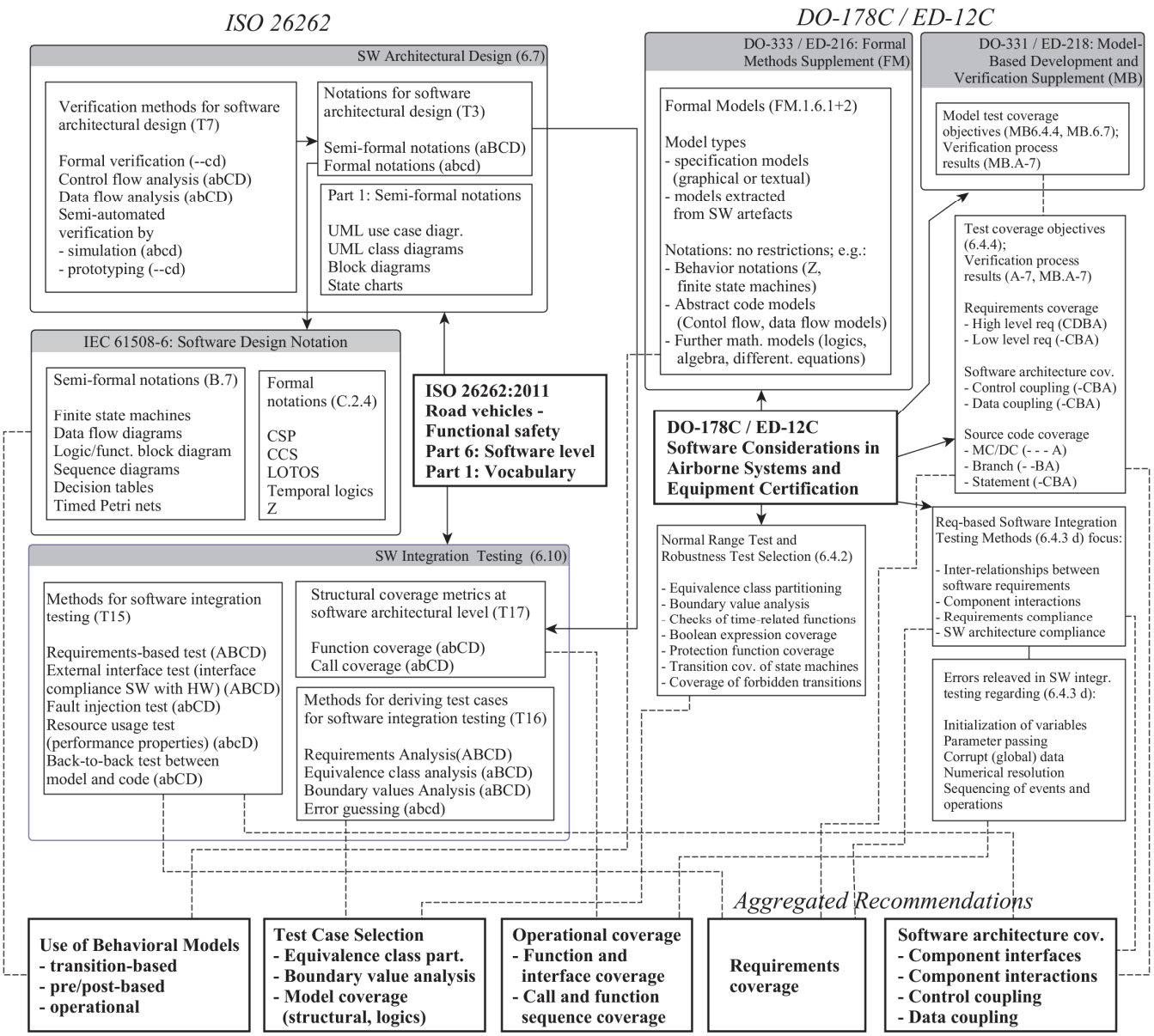


Figure 8. Aggregated recommendations of the ISO 26262 and DO-178C regarding software integration testing

Structural Test Case Generation Based on System Models

Leandro T. Costa, Avelino F. Zorzo, Elder M. Rodrigues, Maicon Bernardino, Flávio M. Oliveira
School of Computer Science - Pontifical Catholic University of Rio Grande do Sul - PUCRS
Porto Alegre, RS, Brazil
Email: leandro.teodoro@acad.pucrs.br, bernardino@acm.org, {elder.rodrigues, flavio.oliveira, avelino.zorzo}@pucrs.br

***Abstract*—Structural testing, or white-box testing, is a technique for generating test cases based on analysis of an application source code. Currently, there are different tools supporting this type of test. However, despite the benefits of these tools, some tasks still have to be performed manually. This makes the test process time consuming and prone to injection of faults. In order to mitigate these problems, this paper presents a Model-based Testing (MBT) approach for deriving structural test cases for different code coverage tools using UML sequence diagrams. Our approach consists of four steps: *Parser*, *Test Case Generator*, *Script Generator* and *Executor*. These steps are based on the four main features of a Software Product Line for MBT tools, from which we derived two automation tools (PletsCoverageJabuti and PletsCoverageEmma) that generate and execute structural test cases, respectively. We also describe a case study, which defines test cases for an application that manages skills of employees.**

***Keywords*—model-based testing; structural testing.**

I. Introduction

The evolution and increased complexity of computer systems have made the testing process an activity as complex as the development process itself. In order to overcome this problem, and to increase the effectiveness in the test case generation process, several tools have been developed to automate software testing. Currently, there are several tools supporting different types of testing, for example, structural testing (or white-box testing), in which the source code of the system is inspected; or, functional testing (black-box testing), in which the functionality of the system is verified. In the last decade, many commercial and academic tools have been developed and used to support testing activities, such as, Java Bytecode Understanding and Testing (JaBUTi) [1], Semantic Designs Test Coverage [2], IBM Rational PurifyPlus [3], EMMA [4], Quick Test Professional [5], EvoSuite [17] or Randoop [15].

However, despite the benefits brought about by these testing tools, it is still necessary to perform several manual or semi-automated activities, for example, to provide test cases or to analyze the test results from running test cases. Furthermore, manual or semi-automated test case generation makes the testing process time consuming and prone to introduction of faults, even by experienced professionals. A solution proposal for this issue is to automate the test case generation process through software testing techniques, such as Model-based Testing (MBT) [6]. This technique consists in the generation of test cases and/or test scripts based on system models, which can include the specification of the characteristics that will be tested. MBT adoption presents several advantages, such as reducing the likelihood of misinterpretation of the system requirements by a test engineer or decreasing of testing time.

Currently, MBT can be used to generate test cases through the use of a wide range of modeling notations, such as Specification and Description Language (SDL) [7] or Unified Modeling Language (UML) [8]. UML provides a notation for modeling some important characteristics of applications, allowing the development of automatic tools for model verification, analysis and code generation.

In this context, this paper presents an MBT approach to drive the automatic generation of test cases and test drivers for measuring test coverage. Our approach uses sequence diagrams to identify the classes/methods under test and to generate test sequences based on the order of execution between the classes and methods described in the sequence diagram. Then, generates strucural test cases with a random test case generation tool, and finally generates test drivers to run the test cases and measure their coverage with the code coverage tools EMMA and JaBUTi. Furthermore, our approach is embedded in a Software Product Line (SPL) and new testing products are generated automatically. Our approach consists of four steps: (a) Parser: extracts test information about the classes and methods to be tested from UML sequence diagrams; (b) Test Case Generator: applies a random test data generation technique to generate an abstract structure, *i.e.*, a text file that describes the test case information in a tool-independent format; (c) Script Generator: generates test scripts/test driver for a specific code coverage tool from the information present in the abstract structure; (d) Executor: represents the test execution for a specific code coverage tool using the test driver generated in the previous step. Although we have applied our approach to object-oriented languages, it is straightforward to apply it to other programming paradigms.

One of the advantages of our approach is related to the reuse of test information, *i.e.*, information described in the abstract structure can be reused to generate test scripts for several code coverage tools, *e.g.*, academic: JaBUTi [1] or EMMA [4]; commercial: Semantic Designs Test Coverage [2] or IBM Rational PurifyPlus [3]. Therefore, a company that is using tool A can, motivated by a technical or managerial decision, easily change to a testing tool B without having to create new test cases. Another advantage is related to the use of UML models to generate test cases. Models provide a representation of the test information at a high level, facilitating the understanding by the test expert responsible for implementing and executing test cases. Moreover, differently from others studies that only describe the process to generate test cases through MBT, our approach is able to instantiate them to generate test drivers that could be executed by different code coverage tools.

Based on our approach, we developed two tools: PletsCoverageJabuti and PletsCoverageEmma. Both tools automatically extract test information from sequence diagrams, generate an

 ISBN: 978-1-61208-367-4

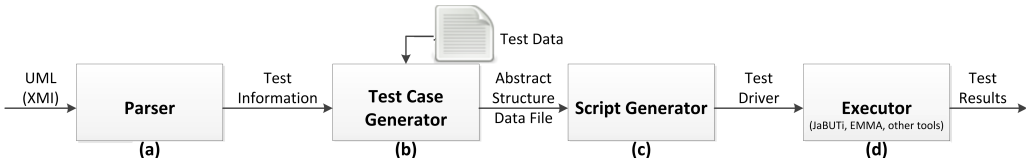


Fig. 1. Approach for generating structural test cases

abstract structure, instantiate the information present in this structure to generate and execute concrete test cases, respectively, for the target tools JaBUTi [1] and EMMA [4]. The tools presented in this paper were derived from a **P**roduct **L**ine for Mod**e**l-based **T**esting tool**s** (PLeTs) [9]. PLeTs supports the generation of products (MBT tools) that automate the generation and execution of test cases. We also applied our approach to a case study, in which we have used two generated tools to test classes and methods of an actual application.

This paper is organized as follows. Section II discusses related background. Section III presents the details of our approach. Section IV describes a case study. Section V discusses related work in structural test case generation using UML. Section VI presents some conclusions and lessons learned.

II. Background

MBT is a technique for automating the generation of test artifacts based on system models [6]. Using MBT it is possible to represent the structure and the system behavior, in order to be shared and reused by the test team members. Therefore, it is possible to extract the test information from models to generate new test artifacts, such as test cases, scripts and scenarios. The MBT adoption requires the creation of models based on system requirements specified by software engineers and test analysts. The purpose is that these models include information that frequently is implicit in traditional specification documents, for example, through comments and/or annotations.

One approach to improve the system specification is the use of UML models [8]. UML models can improve the system specification through stereotypes and tag definitions. Stereotypes is one of the UML extensibility mechanism that may have properties referred to as tag definitions. When a stereotype is applied to a model element, the values of the properties are referred to as tagged values. Hence, all information added to the model, through stereotypes and tagged values, can be used to derive new artifacts, such as test cases.

To the best of our knowledge, early studies focused on MBT were limited to functional testing. Nowadays, models are able to abstract other information, *e.g.*, parameters and input data, thus allowing MBT to be applied to perform other testing techniques, *e.g.*, the structural testing [1].

Structural testing is a technique for generating test cases from the source code analysis. It seeks to evaluate the internal details of implementation, such as test conditions and logical paths. In general, most criteria based on structural analysis use a graph notation named Control Flow Graph (CFG) [1], which represents all the paths that might be traversed during the program execution. These criteria are based on different program elements that can be connected to the control flow and data flow in the program. Control-flow uses the control features of a program to generate test cases, *i.e.*, loops, deviations or conditions, while criteria based on data flow use data flow analysis of the program to generate test cases.

Structural test case generation consists of selecting values from an input domain of a program that satisfies specific criteria. For instance, the All-nodes criterion groups in a domain all the input values that execute a specific node. The selecting input values task could be made using data generation techniques, *e.g.*, random [10], based on symbolic execution [11] or dynamic execution [12]. In this paper, we apply a random technique due to be practical and easier to automate, which provided a useful test case generation for specific code coverage tools.

Currently, there is a diversity of commercial, academic, and open source code coverage tools that assist the testing process. However, most of these tools were individually and independently implemented from scratch based on a single architecture. Thus, they face difficulties of integration, evolution, maintenance and reuse. In order to reduce these difficulties, it would be interesting to have a strategy for automatically generating specific products, *i.e.*, tools that perform tests based on the reuse of assets and a core architecture. This is one of the main ideas behind SPLs [13].

An SPL can be defined as a set of systems that share common and manageable features in order to meet the needs of a specific domain, which may be a market segment or mission [13]. The aim is to explore the similarities among systems in order to manage variability aspects and thus determine a higher reusability level of software artifacts. Through the reuse of artifacts, an SPL allows to create a set of similar systems, thus reducing time to market, cost and, hence, to achieve a higher productivity and quality improvement.

In the testing context, we developed an SPL of MBT tools called PLeTs [9]. This SPL supports the derivation of MBT tools that allow automatic generation and execution of test cases. The purpose of PLeTs is not only to manage the reuse of artifacts and software components, but also to make the development of a new tool easier and faster. Until now, PLeTs was able to generate performance testing products. In this paper, we extend PLeTs to develop structural testing products.

III. Approach to Structural Test Case Generation

As mentioned in the previous sections, MBT techniques have been used to improve software testing through automation of test case generation. Furthermore, using UML models it is possible to automate the test case generation through annotation of test information using stereotypes and tags. Stereotypes and tags can be included in different parts of an UML model to represent test case information [8]. In our previous work [14], we have used UML use cases and activity diagrams as SUT models to automatically generate performance test cases from the information annotated on these diagrams. When conducting performance or even functional testing, UML use cases and activity diagrams were sufficient. However, an understanding about the ordering of execution between program units (*e.g.* methods/functions) is

 ISBN: 978-1-61208-367-4

needed to execute structural testing. In this context, we propose an approach to automate the generation and execution of structural test cases based on UML sequence diagrams. Thus, test sequences are generated according to the order of the methods described in sequence diagrams. As mentioned in Section I, we divided our approach in four steps (see Figure 1): `Parser`, `Test Case Generator`, `Script Generator (Test Driver)`, and `Executor`. These steps are based on the four main features of PLeTs.

In order to generate and execute the *Test Driver*, our approach must retrieve information, about classes and methods, annotated in an UML sequence diagram. It is important to highlight that the diagrams must be well-defined, *i.e.*, they have to contain information about classes and methods parameters (name, type), as well as, each method return type. Besides, it is also necessary to annotate the diagrams with additional information, *e.g.*, a variable that will be used to specify the path of the classes that will be tested. This information will be used to generate the *Abstract Structure* (more details about how the diagram is annotated will be presented in Section IV). The UML sequence diagram is annotated with the following tags: ≪**TDexternalLibrary**≫: specifies the libraries path of the SUT; ≪**TDclassPath**≫: specifies the path of the classes to be tested; ≪**TDtoolPath**≫: specifies information about the chosen code coverage tool, *e.g.*, the installation directory and the path of its launcher; ≪**TDimportList**≫: specifies a list of imported classes.

The advantage of annotating the sequence diagram with these tags is that they are used to provide information used to automatically generate Test Drivers, such as libraries, dependencies among classes and import list. Each tag can define a fixed value or a variable that can be replaced when generating the actual test case or driver for a specific tool. For example, the previously mentioned four tags must be annotated in the sequence diagram with the following parameters: `@externalLibrary`, `@classPath`, `@toolPath` and `@importList`. However, these parameters are just a reference and have no actual information about the code coverage tool, class path, external library or import list. After this annotation process, all information described in the UML sequence diagram is exported to a *XMI* file, which is the input of the first step in our approach.

The first step (`Parser`) consists of parsing the *XMI* file in order to extract the information necessary to generate a data structure in memory, which we call *Test Information* (see Figure 1a). The *Test Information* describes the test sequences generated from the sequence diagram and it has information about the methods and classes to be tested. The second step (`Test Case Generator`) receives as input the *Test Information* and a XML file called *Test Data* (Figure 1b).

The *Test Data* file has the actual values about libraries used to the application execution, the path of classes to be tested and the package list to be imported. However, the *Test Data* file has no tool information, since the first two steps of our approach are tool-independent. Moreover, the *Test Data* also describes a set of different parameter values for all classes and methods of the application to be tested. Based on that, the `Test Case Generator` applies a random test data generation technique [10] under the parameter values presented on *Test Data* and only for the classes and methods described on *Test Information*. The random technique generates input values for each method described in a test sequence. The reason for choosing this technique consists of selecting specific parameters for each one of these classes and methods. It was used due to its practicality and to be easier to automate. However, other techniques are presented in the literature, *e.g.*, symbolic execution [11], dynamic execution [12] and feedback-directed random testing [15]. After applying the random test data generation technique, the `Test Case Generator` also produces the *Abstract Structure* and the *Data File*, which are the input of the third step. The *Abstract Structure* is a text file that describes, in a sequential and tool-independent format, the entire data flow of the classes and methods to be tested (see Figure 3 for an example of file that contains the *Abstract Structure*). The *Abstract Structure* is divided in three groups: **1) Tool Configuration:** defines the `@toolPath` parameter, which specifies the information about the code coverage tool that will be used for the test; **2) Test Configuration:** defines the `@classPath`, `@externalLibrary` and `@importList` parameters, which define the information used for a specific test case; **3) Sequential Flow Configuration:** defines the sequential flow of the methods that will be tested.

Each one of these parameters is a reference to the actual data that is stored in the *Data File*, which is a text file that contains the information (values) used to instantiate test cases for a given code coverage tool (see Figure 4 for an example of a file that contains actual values for a specific tool). The test case instantiation is performed by the step `Script Generator` (see Figure 1c), which consists of automatically generating the *Test Driver* for a specific code coverage tool. Therefore, when the *Abstract Structure* and *Data File* are instantiated to generate *Test Driver*, a class file named *TestDriver.java* is generated. This file contains a class that makes calls to the methods that will be tested and also includes a set of information to be used as input of these methods. In our approach, the input information is generated automatically using the random test data generation technique previously mentioned. Furthermore, in the step `Script Generator` the user must provide all information about a specific code coverage tool, *e.g.*, the path of its launcher.

One of the advantages of using a file to store the actual values, which are used in the instantiation of the class file, is that it is not necessary to include, in the UML sequence diagram, the parameter values of the methods that will be tested. Thus, to generate new test cases with different input values, it is only necessary to generate new test data using any kind of data generation technique. Moreover, the advantage of using the *Abstract Structure* is related to the ability to reuse information for different code coverage tools. In this sense, if a company decides to migrate to a different code coverage tool, due to a management strategy, it will be able to use the test cases previously generated. Besides that, the *Abstract Structure* presents the test information in a clear format, making it simple and easy to understand. Therefore, it is easier to automate the *Test Driver* generation for several tools. The last step (`Executor` - see Figure 1d) consists of performing the test with a specific code coverage tool. Therefore, all the class files generated on step three are used for the test execution. The generation of the class files will be further described in Section IV.

IV. Case Study: Skills - Workforce Planning

This section describes how we have applied our approach to test an application to manage profiles of employees from

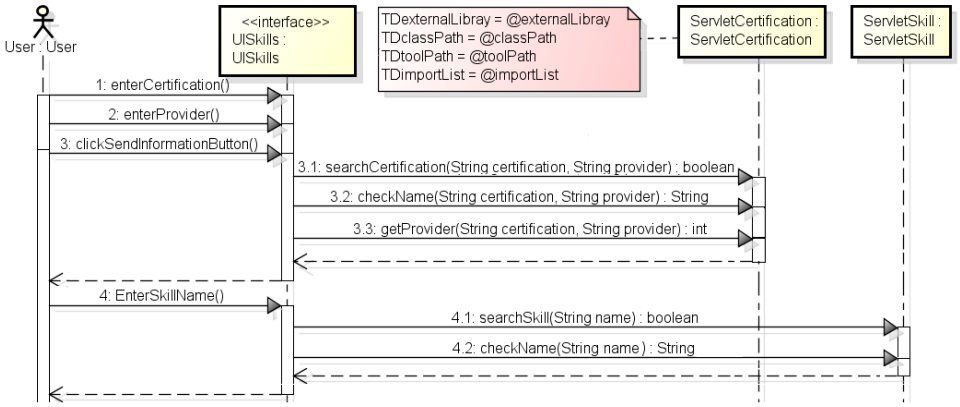


Fig. 2. Sequence diagram

any company. The main goal is to assess the efficacy and the functionality of our approach through presenting how we derived two tools (PletsCoverageJabuti and PletsCoverageEmma) that generate test cases from UML sequence diagrams and execute *TestDrivers* using two coverage tools, *e.g.*, JaBUTi and EMMA. Through the use of our approach we were able to reuse components from steps 1 and 2 (Section III) for both testing tools.

The application used as subject under test is called Skills (Workforce Planning: Skill Management Tool) [14]. This application was developed in a collaboration project between a TDL of a global IT company and our university. The main objective of Skills is to manage and to register skills, certifications and experiences of employees for a given company.

With the purpose of verifying the functional aspects of our approach, we have tested a set of classes and methods of Skills. These classes and methods are represented by four sequence diagrams that describe processes, in which an user performs several operations, *e.g.*: (a) search for a particular certification information; (b) search for a particular skill information; (c) display a list of registered experiences; (d) display information about the user profile; and (e) change the login password. Figure 2 shows part of one of the four sequence diagrams (all sequence diagrams can be found in [16]), in which it is possible to see how tags described in Section III are annotated in the sequence diagram. As can be seen in Table I, these operations are performed through calls of 22 methods of 9 classes (2,561 lines of code). Note that our approach consists of automating the test case generation, in which only the system internal methods are analyzed. Therefore, no method called from the user interaction will be analyzed, since our approach does not implement this feature. In this context, only the information about the methods described in Table I will be used to automatically generate and executing the *Test Driver*.

In order to generate and execute the *Test Driver*, initially, we had to annotate the four sequence diagrams with the tags `TDexternalLibrary`, `TDclassPath`, `TDtoolPath`, `TDimportList` and their respective parameter values: `@externalLibrary`, `@classPath`, `@toolPath` and `@importList`. These tags and values were annotated in the classifier role elements, which represent the nine classes used for this case study. After annotating the sequence diagrams with test information, we exported these test models to a *XMI* file, which is input for PletsCoverageJabuti and PletsCoverageEmma. During their execution, the tools parse the *XMI* file,

```
Abstract Structure: Search for Certification
## Tool Configuration ##
Tool Information : <<TDtoolPath: @toolPath>>
## Test Configuration ##
External Libraries : <<TDexternalLibrary: @externalLibrary>>
Path Classes : <<TDclassPath: @classPath>>
Imported Classes : <<TDimportList: @importList>>
## Sequential Flow Configuration ##
1. ServletCertification
1.1. searchCertification(String certification, String provider):
boolean
1.2. checkName(String certification, String provider): String
1.3. getProvider(String certification, String provider): int ...
```

Fig. 3. Code snippet of the *Abstract Structure*

```
@toolPath = C:\Jabuti\bin; C:\Jabuti\lib\bcel-5.2.jar;
C:\Jabuti\lib\capi.jar; ...
@externalLibrary = C:\Tomcat 6.0\lib\jsp-api.jar; ...
@classPath = C:\CmTool_SkillsTest\web\WEB-INF\classes; ...
@importList = servlets.*; java.io.*; java.util.StringTokenizer
1. ServletCertification
1.1. searchCertification("ActiveX", "BrainBench")
1.2. checkName("ActiveX", "BrainBench")
1.3. getProvider("ActiveX", "BrainBench") ...
```

Fig. 4. Code snippet of the *Data File* for JaBUTi

extracting information from the methods and classes that will be tested in order to generate a data structure in memory (*Test Information*). Based on the *Test Information* and the *Test Data* (a XML file with different parameter values for all classes and methods of the SUT), the tools apply a random test data generation in order to generate the *Abstract Structure* (Figure 3) and *Data File* (Figure 4).

Figure 3 shows a code snippet of the *Abstract Structure* that is divided into three information groups: `Tool Configuration`, `Test Configuration` and `Sequential Flow Configuration`. As mentioned in Section III, all parameters present in each information group are a reference to the actual data that is stored in the *Data File* that contains all values that will be used to instantiate test cases for a given code coverage tool (JaBUTi or EMMA). Figure 4 presents a code snippet with information regarding the parameters values of this file. In this example we defined information on the JaBUTi launcher path (`@toolPath`); for EMMA, we just need to change this value in the *Data File*.

Based on the information described in the *Abstract Structure* and *Data File*, the *TestDriver.java* class is generated. This class is the same for both JaBUTi and EMMA. Since JaBUTi and EMMA perform structural analysis on the bytecode, PletsCoverageJabuti and PletsCoverageEmma create a Java

TABLE I. Coverage Information for JaBUTi

Classes	Methods	Lines of Code	Coverage Percentage (%)			
			One run	*Two runs*	*Three runs*	*Four runs*
ServletCertification	searchCertification	128	100	100	100	100
	checkName	96	100	100	100	100
	getProvider	134	69	75	89	100
ServletSkill	searchSkill	119	56	100	100	100
	checkName	90	100	100	100	100
ServletExperience	getUserExperiences	125	100	100	100	100
ServletProfile	getUsers	122	80	85	93	100
	printResult	95	100	100	100	100
Servlet Password	checkPassword	129	100	100	100	100
	changePassword	121	90	95	100	100
ServletTree	searchSkillNode	120	100	100	100	100
	searchCertificationNode	126	59	72	95	100
ServletIndustryDomain	getRoleChildren	115	48	57	81	100
ServletForgot Password	sendEmail	137	100	100	100	100
	checkEmail	121	66	84	94	100
	checkUser	119	48	63	86	100
ServletGeralSearch	getSelectedUsersCertifications	122	25	50	75	100
	getSelectedUsersExperiences	129	71	82	100	100
	getSelectedUsersSkills	113	74	89	100	100
	printCertifications	100	75	100	100	100
	printExperiences	102	62	100	100	100
	printSkills	98	90	100	100	100

process to compile the driver class. In order to perform test cases with EMMA, automating the generation of the *TestDriver.java* class is enough. However, in order to perform test cases with JaBUTi it is necessary to generate a project file. PletsCoverageJabuti generates this project file by creating a Java process. This process runs a JaBUTi's internal class called `br.jabuti.cmdtool.CreateProject`, in which some information such as paths of the JaBUTi's internal libraries is used as input parameter.

Once these two files are generated, the test execution consists in the internal call of the `probe.DefaultProber.probe` and `probe.DefaultProber.dump` methods for JaBUTi. At the end, the PletsCoverageJabuti creates a Java process for running JaBUTi, which is responsible to calculate and to show the updated coverage information for the defined test case. Based on that converage information, the tester could continue running the PletsCoverageJabuti in order to generate more test cases and increase code coverage. In this context, the tool executes several tests until the code coverage is reached. The tester has also the possibility of terminating the PletsCoverageJabuti execution in any moment and then, finalize the test. An advantage of using PletsCoverageJabuti is that it could generate several tests avoiding redundant test cases, since each test case generated by the random technique is saved by the tool. This ensures that a test case will not be repeated. Table I shows the coverage results after four test runs. It is important to mention that all classes and methods were analyzed based on All-nodes criterion. As can be seen in the table, some methods were covered after one run, while others needed for runs to be covered.

In order to generate and execute *Test Drivers* using the PletsCoverageEmma, we have used the same sequence diagram. However, we have not annotated it with test information, because this task had been done previously for PletsCoverageJabuti. Furthermore, all test cases generated for PletsCoverageJabuti were also used for our second tool. In the same way as PletsCoverageJabuti, the user/tester has the possibility of continuing to run the tool in order to generate and execute more *Test Drivers*. The results for EMMA are similar to the ones for JaBUTi presented in Table I.

These results show that our approach allowed the same diagrams, and test cases to be used in different tools producing similar results. Furthermore, our approach was able to generate a second tool (PletsCoverageEmma) with less effort. The reason is that our approach is based on an SPL, which allowed the reuse of components (*Parser* and *Test Case Generator*) already developed. Although we have developed different components (*Script Generator* and *Executor*) for our both tools, this task required less effort compared to development of the two first components. In this case, we had to automate the calls of internal routines and subcommands of JaBUTi and EMMA. Furthermore, once familiar with the functional features of the PletsCoverageJabuti tool, it was possible to perform tests with little learning effort using PletsCoverageEmma, since both tools share several features, *e.g.*, GUI, test data generation technique and the *Abstract Structure* format.

The results also show the importance of performing structural testing, since it covers faults that are difficult to meet with other testing techniques, *e.g.*, the functional testing. For instance, if a test team does not ensure that all methods were fully covered during the structural testing activity, it is possible that when applying the functional testing, a specific functionality cannot be assessed (unreachable statement) due to a code inconsistency, *e.g.*, infinite loops or conditions that never occur. Therefore, structural testing is useful in combination with functional testing, since it helps to reveal faults that may not be evident with black-box testing alone.

V. Related Work

There has been some work in the past years related to MBT, UML and structural testing, but to the best of our knowledge none of them has integrated all of them. Furthermore, our work also uses code coverage tools and it is integrated into an SPL.

Regarding test case generation using UML sequence diagrams, Khandai *et al.* [18] propose an approach for generating test cases for concurrent systems using sequence diagrams. Our approach, on the other hand, aims to generate tools that automatically generate and execute test cases based on source code of applications. A strategy similar to Khandai *et al.* can be applied to extend our approach for concurrent systems.

Similarly, Sharma *et al.* [19] convert the UML sequence diagram into Sequence Diagram Graph (SDG), and then traverse the SDG to generate the test cases. Other UML diagrams are also used to collect information that is stored in the SDG. Their approach was extended to combine sequence and use case diagrams to generate system test cases. This extended approach consists of converting UML use cases into a Use Case Diagram Graph and UML sequence diagrams into SDG. Our work, on the other hand, focus on structural testing with coverage criteria based on commands, decisions, classes, and methods, which is not addressed by Sharma *et al.*. Thus, both approaches are complementary since our approach can be used to generate test cases to cover interactions and scenarios faults.

A work from Swain and Mohapatra [20] uses UML sequence and activity diagrams to generate test cases by converting the UML diagrams into an intermediate representation called Model Flow Graph (MFG). This MFG is traversed to generate test sequences that are instrumented in the test case to satisfy a message-activity path test adequacy criteria. Our approach differs from [20] since it is embedded in an SPL and go further than only generating the test cases, *i.e.*, our approach actually executes the test cases in two code coverage tools.

Different from the existing works in test case generation presented in this section, our approach consists not only in the test case generation through MBT, but also on the generation and execution of *Test Drivers* for several tools.

 ISBN: 978-1-61208-367-4

Furthermore, we applied our approach to a detailed case study in an actual company environment. Moreover, our approach is based on an SPL, which make it easier to reuse code that was not developed for a specific tool. This has happened in the two tools we presented and also in previous tools for performance and functional testing [14]. Furthermore, our approach distinguishes from [14] in two aspects: first, our approach generates test cases from UML sequence diagrams, while their work uses UML use cases and activity diagrams; second, our approach uses a random test data generation technique to select input values from a specific domain, while their work uses sequence test case generation methods, *e.g.*, Harmonized State Identification (HSI) [21].

VI. Conclusion and Lessons Learned

This paper presented an approach for automating test case generation for several coverage tools from UML sequence diagrams. Based on this approach, we incremented an SPL called PLeTs. PLeTs is able to generate testing tools that use academic or commercial tools to execute performance, functional or structural test. One of the advantages of our approach is related to reuse of test information described in UML sequence diagrams. Hence, it is possible to easily migrate to a different testing tool and reuse the test cases previously defined. The tools used to exemplify our approach were: JaBUTi and EMMA. However, commercial tools such as Semantic Designs Test Coverage and Rational PurifyPlus or other academic tools such as Poke-Tool (Poke-Tool) could be used for this purpose. Furthermore, our approach is useful for industry when developers already have designed models. In this context, the models could be reused to automatically generate test cases. Basically, the main lessons we have learned were: **1. Coverage analysis based on bytecode and source code.** Although we have presented a case study, in which we used two code coverage tools (JaBUTi and EMMA) to perform coverage analysis based on bytecode, our approach is able to deal with tools performing tests based on the analysis of source code, *e.g.*, Semantic Designs Test Coverage [2]. Some minor tools related changes should be performed, however. For example, the `@classPath` parameter must indicate the path of source code files instead of the path of class files (bytecode). We decided to use bytecodes, since in some situations the source code could not be available to test an application. **2. The choice of test data generation technique.** When performing structural testing, it is very important to choose an efficient technique for generating testing data. An efficient technique increases the likelihood of meeting the requirements of structural testing. In our approach we have applied a random testing data generation technique to select the parameter values used to instantiate the *TestDriver.java* file. However, our approach could implement other data generation techniques, *e.g.* symbolic execution [11] and dynamic execution [12]. These two techniques are more effective than random generation and guarantee data selection with a higher probability to reveal faults. Nevertheless, a random technique is practical and easier to automate. **3. The needed knowledge on the code coverage tools.** Although there are several ways to automate the generation and execution of tests for different tools, a detailed study of used code coverage tools is still necessary. Sometimes this study may reveal that is not possible to automate the generation and execution of test cases for a particular code coverage tool. For example, open source tools such as JaBUTi and EMMA are easier to automate than commercial ones, because it is possible to get access to their internal functioning. Another point is related to the way tools are executed, *i.e.*, throughout command line or GUI. Command line tools are easier to automate because they, usually, provide a set of subroutines/programs that can be easily parameterized. **4. The advantage to generate testing tools from an SPL.** The SPL concepts were useful to develop testing tools with less effort. A reason for that is related to the possibility to reuse components already developed to generate other testing tools. Furthermore, an SPL can provide other advantages, such as: quality improvement, since it is possible to reuse components already developed and tested; higher productivity, since it is not necessary to develop tools from scratch; and cost reduction, since it is possible to develop tools in large scale.

Acknowledgment

Research projects: PDTI 001/2014 financed by Dell Computers with resources of Law 8.248/91, and AutoScene supported by Facin/PUCRS. Thanks also to Soraia R. Musse.

References

[1] A. M. R. Vincenzi, M. E. Delamaro, J. C. Maldonado, and W. E. Wong, "Establishing structural testing criteria for Java bytecode," Software: Practice and Experience, vol. 36, no. 14, pp. 1513–1541, 2006.

[2] Semantic Designs, "Semantic Designs Test Coverage," URL: http://www.semdesigns.com, [retrieved: July, 2014].

[3] IBM, "IBM Rational PurifyPlus," URL: http://www.ibm.com/software/awdtools/purifyplus/, [retrieved: July, 2014].

[4] V. Roubtsov, "EMMA: a Free Java Code Coverage Tool," URL: http://emma.sourceforge.net, [retrieved: July, 2014].

[5] S. R. Mallepally, QuickTest Professional (QTP) Interview Questions and Guidelines: A Quick Reference Guide to QuickTest Professional. Parishta, 2009.

[6] P. Krishnan, "Uniform Descriptions for Model Based Testing," in *Proc. ASWEC*, 2004, pp. 96–105.

[7] A. Kerbrat, T. Jéron, and R. Groz, "Automated Test Generation from SDL Specifications," in Proc. SDL Forum, 1999, pp. 135–152.

[8] G. Booch, J. Rumbaugh, and I. Jacobson, The Unified Modeling Language User Guide. Addison-Wesley Professional, 2005.

[9] CePES/PUCRS, "PLeTs SPL," URL: http://www.cepes.pucrs.br/plets, [retrieved: July, 2014].

[10] D. Hamlet and R. Taylor, "Partition Testing does not Inspire Confidence," IEEE Transactions on Software Engineering, vol. 16, no. 12, pp. 1402–1411, 1990.

[11] M. Lin, Y. Chen, K. Yu, and G. Wu, "Lazy Symbolic Execution for Test Data Generation," IET Software, vol. 5, no. 2, pp. 132–141, 2011.

[12] R. Dara, *et al.*, "Using Dynamic Execution Data to Generate Test Cases," in Proc. ICSM, 2009, pp. 433–436.

[13] P. Clements and L. Northrop, Software Product Lines: Practices and Patterns. Addison-Wesley Longman Publishing, 2001.

[14] M. B. Silveira, *et al.*, "Generation of Scripts for Performance Testing Based on UML Models," in Proc. SEKE, 2011, pp. 258–563.

[15] C. Pacheco, S. K. Lahiri, M. D. Ernst, and T. Ball, "Feedback-Directed Random Test Generation," in Proc. ICSE, 2007, pp. 75–84.

[16] CePES/PUCRS, "PLeTs Guide," URL: http://www.cepes.pucrs.br/plets/?a=guide, [retrieved: July, 2014].

[17] EvoSuite, "EvoSuite," URL: http://www.evosuite.org, [retrieved: July, 2014].

[18] M. Khandai, A. Acharya, and D. Mohapatra, "A Novel Approach of Test Case Generation for Concurrent Systems Using UML Sequence Diagram," in Proc. ICECT, 2011, pp. 157–161.

[19] M. Sarma, D. Kundu, and R. Mall, "Automatic Test Case Generation from UML Sequence Diagram," in Proc. ADCOM, 2007, pp. 60–67.

[20] S. K. Swain and D. P. Mohapatra, "Test Case Generation from Behavioral UML Models," International Journal of Computer Applications, vol. 6, no. 8, pp. 5–11, 2010.

[21] A. Petrenko, *et al.*, "Nondeterministic State Machines in Protocol Conformance Testing," in Proc. PTS, 1993, pp. 363–378.

 ISBN: 978-1-61208-367-4

Towards a Maturity Model in Software Testing Automation

Ana Paula C. C. Furtado, Silvio R. L. Meira
Informatics Center – Cin
Federal University of Pernambuco – UFPE
Recife, PE, Brazil
{apccf, srlm}@cin.ufpe.br

Marcos Wanderley Gomes
SOFTEXRECIFE
Recife, PE, Brazil
marcos@recife.softex.br

***Abstract*—The practice of testing software is one of the ways to produce software with quality for demanding clients in the software market. The automation of Software testing may be seen as a solution for how to test the greatest amount of software within a project, due to the fact that the more the software is built, the larger the scope of testing is. Therefore, organizations that seek to guarantee that their software projects are being built according to the demands of their clients should follow an automation approach to testing. Thus, this paper puts forward a description of work in progress on the development of a maturity model for automating software testing that is being developed as part of a doctoral thesis. Besides presenting the overall expected structure of the maturity model, the plan for validating it is also set out.**

Keywords-software testing; automation; maturity models.

I. INTRODUCTION

Software Testing is an essential activity in today´s world of software development, given that customers are more and more rigorous about the quality of products being delivered to the market. It is necessary to test in order to minimize the risks of finding faults in the software while in clients' production environment.

Within this context, automating software testing appears as an alternative to manual tests in order to cover a broader scope of the functionalities tested within a shorter period of time. According to International Software Testing Qualification Board (ISTQB) [1], test automation is the use of software to execute or support testing activities, such as test management, test case, test execution and assessing results. Nevertheless, the automation of testing is an activity that can be introduced in order to gain productivity from the team and additional quality in the artifacts generated.

Another relevant perspective from which to approach excellence in software development is to use maturity models to support the continuous improvement of the processes within an organization. There are maturity models that cover the entire scope of development activities, such as Capability Maturity Model Integration for Development (CMMI-DEV) [2] and MPS.BR [3] (the acronym in Portuguese for Improving Software Processing: a Brazilian model) which is a Brazilian model that was developed with a view to the global software community considering it better suited to its needs. Nevertheless, there are three other models that were built specifically to build more discipline into testing, namely Testing Maturity Model – TMM [4], Test Maturity Model Integration – TMMI [5] and MPT.BR [6] (the acronym in Portuguese for Improving Test Processing: a Brazilian model), and thereby to support the introduction of testing in a more disciplined and prescribed manner.

However, it is observed that none of them discuss testing automation as an issue within maturity models. Organizations that seek to automate their testing have no support from maturity models which would help them to understand what the best practices of automation are and how to introduce these into their organizations.

Therefore, this paper sets out the overall structure of a maturity model for automating software testing that is being developed as part of a doctoral research study.

This paper is organized as follows: the next section gives an overview of the discipline of software testing and its main concepts. Section 3 gives the background to maturity models and comments on what they offer in terms of automating software testing. Section 4 explains the framework for the maturity model and Section 5 makes concluding remarks and suggests future lines of study.

II. SOFTWARE TESTING BACKGROUND

According to ISO/IEEE [7], testing is a set of activities conducted to facilitate discovery and/or evaluation of properties of one or more test items. Testing activities can include planning, preparation, execution, reporting, and management activities, insofar as they are directed towards testing.

Meyers [8] states that software testing is the process of executing a program with the intent of finding errors. The book, *A Guide to Advanced Software Testing* [9], states that testing can also be considered a support activity: it is meaningless without the development processes and does not produce anything in its own right: nothing developed entails nothing to test.

All such statements give a general idea of the definition of software testing and essentially lead to the same overall objective of software testing which is not to find every system/software bug that exists, but rather to uncover situations that could negatively impact the business. Nevertheless, note that the cost of finding and fixing bugs can rise considerably during the development life cycle. Therefore, the earlier in testing that bugs which are judged

 ISBN: 978-1-61208-367-4

likely to have moderate or serious impacts on later stages are identified and fixed, the better.

On the other hand, ISTQB [1] declares test automation as the use of software to perform or support test activities, e.g., test management, test design, test execution and results checking. According to ISO/IEEE [7], automated testing is often considered to be mainly concerned with conducting tests on scripted tests rather than having testers conduct tests manually. However, many additional test tasks and activities can be supported by software-based tools.

The activity of automating tests assumes that tools are used, and, according to Hass [9], the purpose of using tools for testing is to get as many as possible of the noncreative, repetitive, and boring parts of the test activities automated. The purpose is also to exploit the possibility of tools for storing and arranging large amounts of data.

Automation may help solve problems, especially those caused by:

- Work that is to be repeated many times;
- Work that it is slower to do manually; and
- Work that it is safer to do with a tool.

Another goal when introducing automation techniques into the discipline of testing is to increase the productivity of the team. Otherwise the cost of introducing automated practices would not be compensated for. Figure 1 is a graphical representation of the comparison of the cost of manual and automated testing.

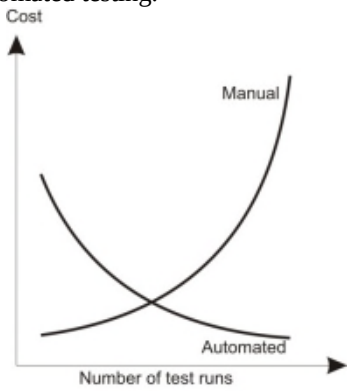


Figure 1. The cost of testing, by Hass [9].

Therefore, this section presented the main concepts of software testing used for this research. The next section comments on maturity model concepts used as references to implement a testing maturity model in automation.

III. SOFTWARE MATURITY MODELS

According to Prado [10], maturity can be defined as "a way to measure the stage that an organization is at in its ability to manage its projects." The main objective is to help improve the way software is being built.

In order to suggest a maturity level for automated testing, the main maturity models studied were CMMI-DEV [2], TMMI [5] and MPT.BR [6], which will be explained in the following sections.

A. TMMI

TMMI [5] is a maturity model that was produced by the TMMI Foundation which used TMM [4] as a reference. It aims to work as a complementary model to CMMI-DEV [2] and, therefore, it is organized in maturity levels, as presented in Figure 2.

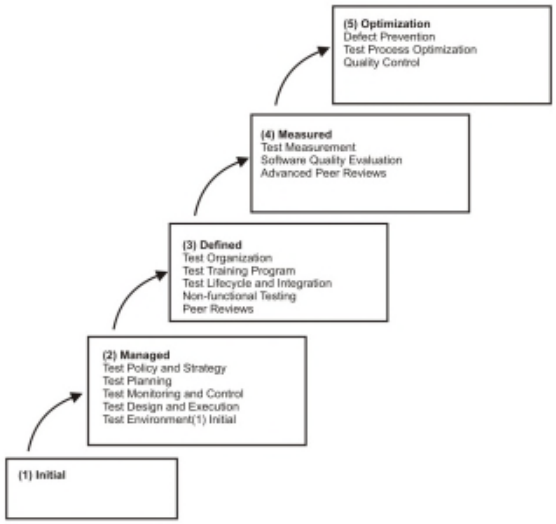


Figure 2. TMMI Maturity Levels [5].

The framework of the model consists of 16 process areas. However, the model states “TMMI does not have a specific process area dedicated to test tools and/or test automation”. The model comments that test tools are treated as a supporting resource (practices) and are therefore, part of the process area where they provide support.

B. MPT.BR

MPT.BR [6] approaches the enhancement of the testing process by using the best software testing practices throughout the product lifecycle. MPT.BR uses guidelines on how best to improve the software testing process throughout the lifecycle of the software.

It was developed to be introduced as a complement of MPS.BR [3], which focuses on software processing, but pays scant attention to testing disciplines.

The MPT.BR reference model presents five maturity levels, representing the stages for evolving a test process in the context of an organization. The maturity levels are shown on Table I.

The levels comprise 16 processes areas, one of which, AET (which is the acronym in Portuguese for Test Execution Automation), that specifically addresses testing automation, the objective of which is to establish and maintain a strategy for automating test execution activity, by defining its objective, defining a framework and assessing the Return on Investments (ROI).

There is another process area, called GDF (which stands for tools management), that mentions testing tools. Its objective is to manage the identification, analysis, selection and deployment of tools to support testing activities, in

 ISBN: 978-1-61208-367-4

general, within an organization. This process area does not mention any specific tool; it talks about the necessity to plan organizationally, to instantiate and to manage the use of tools within a project.

TABLE I. MPT.BR PROCESSES AREAS

Maturity Level	Objective
1 – Partially Managed	This contains the minimal requirements that a company needs to meet in order to demonstrate that the discipline of testing is applied to projects and that this takes place in a planned and monitored manner.
2 – Managed	This takes a broader view in which the scope of the project starts to be controlled by the management of change process. In addition, software testing patterns are defined and processes are monitored and controlled.
3 – Defined	At this level, testing becomes organizational. Defined software processes are adopted, quality Assurance is institutionalized in order to support process definition, responsibilities for test organization are defined and a measurement program is institutionalized in the organization. At this level, the software testing lifecycle is associated with the development one, where static and acceptance testing are formalized and systematic procedures are applied for test closure.
4 – Defect Prevention	This focuses on preventing defects and systematically improving the quality of the product. At this level, the organization has a process for managing defects, in which defects found are monitored.
5 – Automation and Optimization	The fifth maturity level sets out to establish a process for testing that continuously improves tests and automates them.

C. CMMI-DEV

CMMI-DEV [2] is a model that consists of best practices that address development activities applied to products and services. It addresses practices that cover the product's lifecycle from conception through delivery and maintenance.

The structure of the model comprises 22 process areas organized in 5 maturity levels, which are:

1. Initial;
2. Managed;
3. Defined;
4. Quantitatively Managed; and
5. Optimizing.

CMMI is a maturity model that can be applied by means of staged or continuous representation. In the former, the organization can improve a set of related processes by incrementally addressing successive sets of process areas. The latter enables organizations to improve processes corresponding to individual process areas, by making it possible to choose the ones that best fit the organizational environment.

Both representations use the same set of process areas, and there are 2 that specifically talk about testing, as shown in Table II.

TABLE II. CMMI PROCESS AREAS OF TESTING

Process Area	Description.
Verification	The purpose of Verification (VER) is to ensure that selected work products meet their specified requirements.
Validation	The purpose of Validation (VAL) is to demonstrate that a product or product component fulfills its intended use when placed in its intended environment.

Both process areas talk about practices on how to guarantee quality by means of testing activities (static and dynamic testing), but there are no recommendations on how to conduct automated practices for testing activities.

D. Automation Approach on Maturity Models

This section combines the maturity models that are used as main references to build the MPTA.BR. Table III summarizes the maturity models together with the approach of automation contained in each, if present.

TABLE III. MATURITY MODELS AND AUTOMATION APPROACH

Maturity Model	Automation Approach
CMMI	No automation approach defined, there are two process areas that talk about testng, namely, VER and VAL.
TMMI	Automation can be done in any process area but there is no guidance on how to do it.
MPT.BR	Level 5 presents two process areas, one of which is AET which talks about automation and GDF which mentions tools, in general, including automated ones.

The next section will present the proposal of the work in progress for developing a maturity model for automation

IV. MPTA.BR

The Maturity Model MPTA.BR (the acronym in Portuguese for Improving the Test Automation Process: a Brazilian model) aims to be complementary to MPT.BR, as it provides guidance to be used when developing automation processes within an organization.

In the current marketplace, maturity models, standards, methodologies, and guidelines exist that can help an organization improve the way it does business. According to CMMI-DEV [2], "the quality of a system or product is highly influenced by the quality of the process used to develop and maintain it."

The idea to develop a maturity model on automation arose from both technical research and a bibliographic review as well as from demand in the software development industry. Research specifically undertaken in organizations that have achieved a maturity level on MPT.BR [11] shows that they are interested in applying automation techniques to their testing processes.

MPTA.BR will follow the structure of MPT.BR, where each maturity level consists of a set of process areas, which can be understood as a group of related practices that, when implemented together, satisfy a specific objective. Each maturity level is also associated with generic practices that are applied to each process area.

 ISBN: 978-1-61208-367-4

A. Maturity Levels

The maturity levels were influenced by the organization of MPT.Br, together with a classification of tools for automation defined by Hass [9]. This describes an evolutionary track recommended for a company that aims to introduce testing automation processes. The maturity levels are:

1. **Managed**: its objective is to introduce the automation of planning and monitoring activities of the test project as well as configuration management practices.
2. **Designed**: This maturity level focuses on the definition of automated practices in test design and debugging activities, as well as troubleshooting and static analysis tools.
3. **Executed:** The objective of this level is to automate data generation, simulation, emulation, fault-sending, fault-injection and test case execution.
4. **Analyzed:** The objective is to use tools to support a comparison of results and indicators.

The suggested model improves the concepts presented in the other models to include specific guidance on how to introduce test automation in an organization.

B. Validation of MPTA.BR

The validation of the model is planned to occur through the following processes of Case Study and Survey.

The case study is run by adopting the model in selected and volunteer organizations following the steps below:

1. Making an initial diagnosis to identify gaps and to assess the automation practices that exist (if any) in the organizations;
2. Building an action plan to introduce the practices of the model in the organization;
3. Running a pilot project; and
4. Assessing the pilot project to identify if the intended objectives of the model were achieved, in fact, with the support of MPTA.BR.

On the other hand, a survey with specialists can be conducted by selecting a group of experienced professionals, both from the academic world and industry. Thereafter, it is necessary to run a survey in order to assess their opinion on the effectiveness of the model with regard to helping to introduce automation practices in organizations.

Both methods of validation can run in parallel and after collecting the results from both, the positive and negative aspects of the model will be assessed and the improvement opportunities consolidated in order to generate the final version of the model which the doctoral research study sets out to do.

Certain limitations to validate the model can be observed, such as the difficulty to find an organization to run the case study and to select/find the correct specialist to execute the survey.

V. CONCLUSION AND FUTURE WORK

It has been observed that software testing automation can be used to support organizations to achieve higher levels of quality in the products being developed by the software industry. Maturity models that are being used world-wide give little, or almost no guidance on how to implement automation in testing processes.

This work in progress is part of a proposal for a doctoral thesis that is being developed and was prompted by prior research and a review of the literature besides which it was noted from personal experience and observations that there is a demand from the software industry for a model of this nature. The objective is to propose guidelines using a maturity model on software testing automation in order to help organizations gradually introduce automation practices.

One of the threats that may arise from this research is related to the fact that automation might not be the solution for an organization´s needs and its introduction may make the process heavier than necessary.

Another relevant threat is that even defining MPTA.BR as the model to be implemented might make it more difficult than expected to run study cases in the real world environment because it is hard to convince organizations to introduce practices of a model that is under construction.

Automation may not be the solution for all software development projects because its incorrect use may lead to an increase in cost and not make sense in the end. As to future research, this model will be detailed with the information necessary and this will include making detailed descriptions of its structure and processes areas.

REFERENCES

[1] ISTQB, “Standard Glossary of Tearms Used in Software Testing”, Version 2.2, October 2012.

[2] CMMI-DEV, “CMMI for Development”, Version 1.3, CMU/SEI-2010-TR-033, Software Engineering Institute, 2010.

[3] MPS.BR, “Improving Software Processing: a Brazilian model ”, Softex, Available from http://www.softex.br/mpsbr/guias/, 2014.08.11

[4] TMM, “Test Maturity Model”, Illinois Institute of Technology, Available from http://science.iit.edu/computer-science/research/testing-maturity-model-tmm 2014.08.11

[5] TMMI, “Test Maturity Model Integration” Release 1.0. TMMi Foundation, Ireland. Available from http://www.tmmi.org/pdf/TMMi.Framework.pdf 2014.08.11.

[6] Softex Recife, “MPT - Improving Test Processing: a Brazilian model”. Available in http://mpt.org.br/mpt/wp-content/uploads/2013/05/MPT_captured on Apr. 27th 2014. Portuguese Version Only.

[7] ISO/IEEE 29119 – Part I International Standard, “Software and Systems Engineering/Software Testing, Concepts and Definitions, First Edition, 2013.

[8] J. Glenford Myers, “The Art of Software Testing,” John Wiley and Sons, ISBN 0-471-04328-1, 1979

[9] A. Hass, “A guide to Advanced software testing,” Artech House, INC.2008

[10] D. Prado, “Project Management in Organizations”, 2 ed. Belo Horizonte: Editora de Desenvolvimento Gerencial, 2003. Portuguese Version Only.

[11] A. Furtado, M. Gomes, E. Andrade, I. de Farias Junior, “MPT.BR: A Brazilian Maturity Model for Testing” The 12th International Conference on Quality Software (QSIC), August 2012, pp. 220-229, ISSN 1550-6002, ISBN: 978-1-4673-2857-9, DOI 10.1109/QSIC.2012.53

 ISBN: 978-1-61208-367-4

Low-Variance Software Reliability Estimation Using Statistical Testing

Fouad ben Nasr Omri, Safa Omri and Ralf Reussner

Chair for Software Design and Quality
Karlsruhe Institute of Technology
Karlsruhe, Germany
Email: {fouad.omri, safa.omri, ralf.reussner}@kit.edu

***Abstract*—Software statistical testing establishes a basis for statistical inference about the expected field quality of software based on an expected operational profile. The standard statistical testing approach draws randomly test cases from the expected operational profile according to the statistical distribution on the expected inputs. Standard Statistical testing is in the most of the cases impractical. The number of required test cases to reach a target reliability with a given confidence is too large. In this paper, we present a test selection approach to minimize the variance of reliability estimator and reduce the overhead of estimating reliability. The presented approach combines the idea of statistical testing with the concepts of stratified sampling. Experiments are conducted to validate the efficiency of our approach.**

Keywords–Software reliability testing, reliability estimation, statistical testing, stratified sampling

I. INTRODUCTION

Statistical testing draws test cases from the expected Operational Profile (OP) according to the statistical distribution on the expected inputs.

Reliability assessment using statistical testing can be grouped in three different categories: (i) reliability growth models, (ii) fault seeding models and (iii) sampling models. Reliability growth models are making assumptions about the number of faults removed at each testing step by trying to extrapolate the future behavior of the software based on its past behavior. The assumptions made by reliability growth models are difficult to justify [1][2]. Fault seeding models are also making assumptions about the distribution of faults remaining in the program after testing. Such assumptions cannot be rigorously justified [3].

One class of reliability assessment approaches using statistical testing are sampling models. These models are theoretically sound [4], but they suffer from several practical problems. Sampling models require a large number of test cases [5]. In addition, a major concern is how to choose a proper estimator for the reliability that provides accurate and stable reliability estimate. The goodness of an estimator is judged based on the following four properties: (i) unbiased, (ii) minimum variance, (iii) consistent and (iv) sufficient. The main focus when selecting an estimator is the minimum variance of the estimator. The other three properties are in most of the cases satisfied by most of the estimators. The variance of an estimator describes the closeness of the future estimate to the previous estimate when rerunning the estimation with the same setting. An estimator with low variance increases the confidence on the predicted estimate. In fact, a low variance usually implies tighter confidence interval for the estimate. Consequently, we can improve the accuracy of the reliability estimation by providing or choosing an unbiased estimator that has a minimum variance. It is also important to note that the more tests are executed the more will the variance of the estimator decrease. Consequently, an estimator with low variance can find an accurate estimation with fewer test cases.

This paper presents a test selection strategy based on adaptive constrained sampling of the OP to deliver an unbiased reliability estimator which is both efficient and accurate (i.e., needs less test cases than standard approaches to find an accurate estimate). We call our approach Adaptive Constrained Test Selection (ACTS).

The rest of the paper is organized as follows. Section II formulates the problem of reliability estimation variance reduction and identifies the adaptive optimal test cases allocation over the operational profile sub-domains as a solution. The main steps of our approach are described in detail, in Section III. Experiments are set up to validate the performance of the proposed approach in Section IV. We give an overview on related work in Section V. Section VI concludes this paper and proposes future research direction.

II. PROBLEM FORMULATION

The target of this paper is to present a reliability estimation approach that minimizes the variance of the reliability estimator for discrete-time domain software. For discrete-time domain systems, one is interested in the probability of success of a single usage of the software.

A. Software Statistical Testing

Software statistical testing is testing based on an operational profile.

Operational Profile Definition: As defined by Musa [6] "an Operational Profile (OP) is a quantitative characterization of how a (software) system will be used". It consists of a set of partitions of the software input domain (sub-domains) and their probabilities of occurrence.

In most of the cases, the OP describes also the distribution of the input variables of a program. We use in this work the abstract OP representation defined by Musa [6], which we introduce in Section II-B.

 ISBN: 978-1-61208-367-4

B. Standard Tests Selection Approach

Statistical Testing as proposed by Musa [6] generates by random sampling test cases according to the OP.

The OP is used to divide the input domain $\mathcal{D}$ of the software to test in L disjoint sub-domains: $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_L$. Each sub-domain represents a possible operational use and has a probability of occurrence according the OP. Let p_i be the probability of occurrence of sub-domain $\mathcal{D}_i$. The OP can be represented as $OP = \{(\mathcal{D}_i, p_i) | i = 1, 2, \ldots, L\}$.

Let $\mathcal{A}$ a sequence defined as follows: $\mathcal{A} = \{\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_L\}$, $|\mathcal{A}| = L + 1$, where $\mathcal{A}_i = \sum_{k=1}^{i} p_i$ for $i = 1, \ldots, L$, and $\mathcal{A}_0 = 0$.

The generation of the test cases is then as follows:

1) Generate an uniformly distributed random number $\zeta \in (0, 1)$, if $\zeta \in [\mathcal{A}_i, \mathcal{A}_{i+1}]$, then the sub-domain $\mathcal{D}_{i+1}$ will be randomly sampled since $\mathcal{A}_{i+1} - \mathcal{A}_i = p_{i+1}$, where p_{i+1} the probability of occurrence of sub-domain $\mathcal{D}_{i+1}$.
2) Generate input variables from the sub-domain $\mathcal{D}_{i+1}$ based on the provided input distributions, and execute the test case.
3) Repeat the above steps until a stopping criteria is reached (e.g, target reliability value reached, target confidence on the estimated reliability reached, required test time reached, etc.)

C. Discussion

The test selection approach proposed by Musa [6] is a random selection process without replacement. The selection is controlled by the uniformly distributed random variable $\zeta \in (0, 1)$. The main idea behind this approach is to ensure that when the testing process is terminated because of (for example) imperative software project constraints, then the most used operations will have received the most testing effort. Musa also claims that "the reliability level will be the maximum that is practically achievable for the given test time" [6]. One key assumption here is that the sample of selected test cases represents the expected software execution according to the OP and delivers the maximum achievable reliability level. However, this assumption is not always valid. It would be ideal if we could separate successful program execution from the failing ones. However, this is not likely, because failures are often caused by small faults in a large program.

A software fault is a hidden programming error in one or more program statements. A program consists of a set of statements. A program execution is a program path executed with an input value from the program's input domain. A program path is a sequence of statements. Each program path has an input and an executed output which usually depends on the input. Consequently, a program execution is considered as a failure if the corresponding executed program path deviates from the expected output.

Two program execution are similar if they execute the same program path with different input value. If the same input value is used then the two executions are equal.

Two similar program executions may differ only in regard to executing a particular fault, with the result that one execution fails while the other does not. Conversely, two dissimilar program execution may both fail because they execute the same faulty program statement. Consequently, we may not group the failing program executions together even if they have the same causing fault.

Hence, it is realistic to assume that the reliability estimate across the test sub-domains have different statistical properties (i.e., mean and variance). In this case, we refer to the sub-domains as heterogeneous sub-domains. Using conventional proportional random sampling to select test cases from heterogeneous sub-domains does not guarantee that a statistically sufficient number of test cases will be selected from every sub-domain. Hence, the statistical quality of the samples may be compromised for some sub-domains. This may lead to inaccurate statistical estimate.

Stratified sampling is designed to cluster a population made of heterogeneous groups into disjoint strata and then randomly sampling each strata. This paper addresses the problem of heterogeneity of the OP sub-domains by using optimal stratified sampling. The goal is to formulate the statistical testing approach as an optimal stratified random sampling process and provide a reliability estimator which should reduce the number of required test cases for the estimation while delivering accurate reliability estimates.

D. Stratified Sampling Variance Reduction

Stratified sampling is based on the idea of iterated expectations [7]. Let Y be a discrete random variable taking values $y_1, y_2, ..., y_L$ with probabilities $p_1, p_2, ..., p_L$. Let X be a discrete random variable. Then, $E[X] = E[E[X|Y]] = \sum_{l=1}^{L} E[X|Y = y_l]p_l$. Suppose that the population can be divided into $L > 1$ groups, known as **strata**. Suppose that a stratum l contains N_l units from the population ($\sum_{l=1}^{L} N_L = N$), and the value for the units in stratum l are $x_{1l}, x_{2l}, ..., x_{N_l l}$.

Let $W_l = \frac{N_l}{N}$ and $\mu_l = \frac{1}{N_l} \sum_{i=1}^{N_l} x_{il}$, then it follows that the population mean is $\mu = \frac{1}{N} \sum_{l=1}^{L} \sum_{i=1}^{N_l} x_{il} = \frac{1}{N} \sum_{l=1}^{L} N_l \mu_l = \sum_{l=1}^{L} W_l \mu_l$.

Then, instead of taking a simple random sample (SRS) of n units from the total population, we can take a SRS of size n_l from each stratum ($\sum_{l=1}^{L} n_l = n$). Here $\mu_l = E[X|stratum\ l]$ and $W_l = P[Stratum\ l]$, so the overall mean satisfies the setup of an iterated expectation.

Let $X_{1l}, X_{2l}, ..., X_{n_l l}$ be a sequence of independent and identically distributed random variables samples from stratum l, $\bar{X}_l = \frac{1}{n_l} \sum_{i=1}^{n_l} X_{il}$ be the sample mean, and ${S_l}^2 = \frac{1}{n_l - 1} \sum_{i=1}^{n_l} (X_{il} - \bar{X}_l)^2$ be the sample variance. Then, an estimate of the population mean μ is: $\bar{X_S} = \sum_{l=1}^{L} \frac{N_l}{N} \bar{X}_l = \sum_{l=1}^{L} W_l \bar{X}_l = \sum_{l=1}^{L} W_l \frac{1}{n_l} \sum_{i=1}^{n_l} X_{il}$. Since the random variables X_l are independent, then:
$\text{var}(\bar{X_S}) = \sum_{l=1}^{L} {W_l}^2 Var(\bar{X}_l) = \sum_{l=1}^{L} {W_l}^2 \frac{1}{n_l} (1 - \frac{n_l - 1}{N_l - 1}) {\sigma_l}^2$,
where $\sigma_l^2 = \frac{1}{N_l} \sum_{i=1}^{N_l} (x_{il} - \mu_l)^2$ is the variance of stratum l.

If we assume that $n_l \ll N_l$ for each stratum l so that the finite population factor $FPC = 1 - \frac{n_l - 1}{N_l - 1} \approx 1$ can be ignored, then:

$$\text{var}(\bar{X_S}) = \sum_{l=1}^{L} {W_l}^2 \frac{1}{n_l} {\sigma_l}^2 = \frac{1}{N} \sum_{l=1}^{L} {W_l}^2 \frac{{\sigma_l}^2}{a_l} \qquad (1)$$

where $a_l = n_l / N$ indicates the fraction of samples drawn from the stratum l.

 ISBN: 978-1-61208-367-4

This variance is controllable through the allocation ratio a_l. For example, the proportional allocation, where $a_l = W_l.N/N = W_l$, yields the variance $\text{var}(\bar{X_S}) = \frac{1}{N}\sum_{l=1}^{L} W_l {\sigma_l}^2$.

By Lagrange multiplier method, the optimal allocation $a^* := (a_1^*, \ldots, a_L^*)$ is derived in closed form

$$a_k^* = \frac{W_k.\sigma_k}{\sum_{l=1}^{L} W_l.\sigma_l} \tag{2}$$

achieving the minimal variance $\text{var}(\bar{X_S}) = \frac{1}{N}\sum_{l=1}^{L} {W_l}^2 \frac{{\sigma_l}^2}{a_l^*} = \frac{1}{N}(\sum_{l=1}^{L} W_l\sigma_l)^2$, [7].

Moreover, due to the mutual independence of samples across the strata, the empirical mean $\bar{X_S}$ is asymptotically normal [7].

E. Assumptions

In order to formulate the concerned research goal, some assumptions on the software are presented.

1) The software is frozen when estimating the reliability, since reliability estimation aims at testing the current status of the software. The software will not be modified during the estimation process. The software can be modified after the estimation process.
2) The output of each test is independent of the testing history. In some cases, it is possible that a test case is judged to be failure free although it actually leads to some faults which cannot be observed due to limited test oracles. We consider such test cases to be failure free. However, such unobserved faulty program states can cause the failure of some following test cases. Consequently, the latter test cases can be mistakenly considered as faulty test cases. This leads to an error in the reliability estimation. However, this is not a reliability estimation approach concern rather is a test oracle problem.
3) Each test case either fails or succeeds. A test oracle is used to verify the behavior of the software under test.
4) We assume that a proper test oracle is available, since this work focuses on the effectiveness and efficiency of reliability estimation.
5) In each operational use represented by a sub-domain $\mathcal{D}_i$, all possible software operations and possible inputs are equally likely to arise.
6) We assume that an OP is provided for the tested software.

III. Adaptive Constrained Test Selection

The $OP = \{(D_i, p_i) | i \in \{1, ..., L\}, \sum_{i=1}^{L} p_i = 1\}$ defines the expected input domain of the program's input variables. Each partition (D_l, p_l) is a subset of the OP, and $p_l \geq 0$ is the probability that a program input belongs to sub-domain D_l. The OP is a natural definition of the strata for stratified random sampling. Each stratum l corresponds to the sub-domain $\mathcal{D}_l$ and has a weight $W_i = p_l$.

A test case either fails or not. Consequently, each test case execution is a Bernoulli trial. Let X_{il} be the outcome of test case i from stratum l, i.e., from sub-domain $\mathcal{D}_l$, then:

$$X_{il} = \begin{cases} 1, & \text{if the test case fails} \\ 0, & \text{if the test case does not fail} \end{cases}.$$

Let μ_i defined as $P(\text{test cases from sub-domain } \mathcal{D}_i \text{ fail}) = \mu_i$, where $i = \{1, 2, \ldots, L\}$ and $\mu_i \in [0, 1]$.

Based on assumption 2, $\{X_{il}\}$ are independent random variables, and since $\sum_{i=1}^{L} p_i = 1$, then it can inferred that $P(X_{il} = 1) = \mu_i$ (i.e., the probability that test case i from sub-domain $\mathcal{D}_l$ fails). Each test case will lead the software under test to failure or success. And in each sub-domain the probability of failure of each test case is equal for all test cases in the sub-domain. Hence, the distribution of X_{il} is binomial distribution with μ_i.

Consequently, the sample mean of stratum l, $\bar{X_l} = \frac{1}{n_l}\sum_{i=1}^{n_l} X_{il}$ is an unbiased point estimator of μ_i.

The reliability of the tested software can be defined as the weighted sum of the reliability of the sampled OP sub-domains $\mathcal{D}_{i,i=\{1,\ldots,L\}} : R = \sum_{i=1}^{L} p_i(1 - \mu_i)$. An unbiased estimator of the reliability is then defined as:

$$\widehat{R} = 1 - \sum_{i=1}^{L} p_i \bar{X}_i = 1 - \sum_{l=1}^{L} \frac{1}{n_l}.p_l \sum_{i=1}^{n_l} X_{il} \tag{3}$$

with $E[\widehat{R}] = 1 - \sum_{i=1}^{L} p_i\mu_i$ and $\text{var}[\widehat{R}] = \sum_{i=1}^{L} {p_i}^2 \frac{\mu_i.(1-\mu_i)}{n_i} = \sum_{i=1}^{L} {p_i}^2 \frac{{\sigma_i}^2}{n_i}$, since the distribution of X_{il} is a binomial distribution with μ_i.

A. Optimal Test Cases Selection

The Problem of selecting the test cases optimally from the OP sub-domains is an adaptive optimization problem formulated as follows. Given the OP, we want to select a total number n of test cases, where (i) n_i test cases are selected from each sub-domains $\mathcal{D}_{i,i\in\{1,\ldots,L\}}$ and (ii) $\sum_{i=1}^{L} n_i = n$, with the goal to minimize $\text{var}[\hat{R}]$. For mathematical tractability, we assume in this section that the total number of required test case n as well as the sub-domains failure rates and consequently their variances are known. These assumptions will be relaxed in the next sections. According to Section II-D:

$$n_i = n\frac{p_i\sigma_i}{\sum_{k=1}^{L} p_k\sigma_k} \tag{4}$$

Note that the larger the variance ${\sigma_i}^2$ of the failure rate of the software when executed with inputs from the sub-domain $\mathcal{D}_i$, the more test cases should be selected from that sub-domain. This makes sense, since the sub-domain with higher estimated/observed failure rate variability should require more testing to attain the same degree of precision as those with lower variability. If the variances of all sub-domains are all equal, the optimal allocation is proportional allocation.

B. Constrained optimal allocation

The intuition behind statistical testing is that the highest the probability of occurrence of a sub-domain, the larger the number of test cases executed from that sub-domain.

To account for this, the optimal allocation introduced in the previous section is formulated as a constrained optimization to a utility cost function c^*. Let $c_i = 1 - p_i$ the cost of selecting

 ISBN: 978-1-61208-367-4

1: **if** $\mathcal{SC}(\mathcal{T}_{OP}) \neq 1 \wedge \mathcal{SC}_{min} = \mathcal{SC}(\mathcal{T}_{(\mathcal{D}_k,p_k)}) < 0$ **then**
// $\mathcal{T}_{(\mathcal{D}_k,p_k)}$ is over-proportional sampled
2: $n = \lceil \frac{n_k}{p_k} \rceil$
3: **for** $\mathcal{T}_{(\mathcal{D}_i,p_i)} \in \mathcal{T} \wedge \mathcal{T}_{(\mathcal{D}_i,p_i)} \neq \mathcal{T}_{(\mathcal{D}_k,p_k)}$ **do**
4: $n_i = \lceil n.p_i \rceil$
5: //select extra $(\lceil n.p_i \rceil - n_i)$ test cases
6: **end for**
7: **else**
8: **for** $\mathcal{T}_{(\mathcal{D}_i,p_i)} \in \mathcal{T}$ **do**
9: $n_i = \lceil n.p_i \rceil$
10: **end for**
11: **end if**

Figure 1. Adjust to Proportional Sampling

a test case from a sub-domain $\mathcal{D}_i$ that has a probability of occurrence p_i, Then

$$n_i = c^* . \frac{p_i \sigma_i / \sqrt{c_i}}{\sum_{k=1}^{L} p_k \sigma_k / \sqrt{c_k}} \quad (5)$$

The cost function c^* is defined in Section III-D.

Note that the higher the cost c_i of selecting a test case from sub-domain $\mathcal{D}_i$, the smaller the sub-domain sample size n_i.

Since the cost function c_i is defined as $c_i = 1 - p_i$, then (5) means: the smaller the probability of occurrence of a sub-domain $\mathcal{D}_i$, the smaller the sample size n_i.

C. Similarity Confidence

When testing a software according to an OP, the goal is to simulate the expected software execution as described by the OP. Consequently, it is interesting to quantify the similarity of the total set of selected test cases to the expected OP. It is also interesting to control the testing process toward a 100% similarity to the OP.

Let $\mathcal{T}_{(\mathcal{D}_i,p_i)}$ be the set of test cases selected from the sub-domain $(\mathcal{D}_i,p_i)_{\{i\in 1,...,L\}}$. Let $|\mathcal{T}_{(\mathcal{D}_i,p_i)}| = n_i$, i.e., the set $\mathcal{T}_{(\mathcal{D}_i,p_i)}$ contains n_i different test cases selected from the sub-domain $(\mathcal{D}_i,p_i)_{\{i\in 1,...,L\}}$. Let $\mathcal{T}_{OP} = \{\mathcal{T}_{(\mathcal{D}_i,p_i)} | (\mathcal{D}_i,p_i) \in OP = \{(\mathcal{D}_i,p_i) | i \in \{1,...,L\}, \sum_{i=1}^{L} p_i = 1\}\}$ the set of selected test cases from the OP. The similarity of $\mathcal{T}_{(\mathcal{D}_i,p_i)}$ to the OP when a total number $n = |\bigcup_{(\mathcal{D}_i,p_i)\in OP} \mathcal{T}_{(\mathcal{D}_i,p_i)}| = |\mathcal{T}_{OP}|$ of test cases is selected from the OP sub-domains, is defined as follows:

$$\mathcal{SC}(\mathcal{T}_{(\mathcal{D}_i,p_i)}) = \begin{cases} \frac{n_i}{\lceil p_i.n \rceil}, & \text{if } n_i \leq \lceil p_i.n \rceil \\ -\frac{n_i}{\lceil p_i.n \rceil}, & \text{if } n_i > \lceil p_i.n \rceil \end{cases} \quad (6)$$

The similarity confidence of the total selected test cases is consequently defined as follows: $\mathcal{SC}(\bigcup_{(\mathcal{D}_i,p_i)\in OP} \mathcal{T}_{(\mathcal{D}_i,p_i)}) = \frac{\sum_{i=1}^{L} \mathcal{SC}(\mathcal{T}_{(\mathcal{D}_i,p_i)})}{L}$

Let $\mathcal{SC}_{min} = min\{\mathcal{SC}(\mathcal{T}_{(\mathcal{D}_i,p_i)}) | i \in \{1,...,L\}\} = \mathcal{SC}(\mathcal{T}_{(\mathcal{D}_k,p_k)})_{k\in\{1,...,L\}}$, the minimum computed similarity to the OP.

Algorithm 1, adjusts the allocation of the test cases from each sub-domain $(\mathcal{D}_i,p_i)_{\{i\in 1,...,L\}}$ to reach a similarity confidence of 100%. The steps of the algorithm are as follows. If the selected tested cases $\mathcal{T}_{OP}$ is not similar to the OP and if $\mathcal{SC}_{min} = SC((D_k,p_k))$ is negative (line 1), then it means that the sub-domain $(\mathcal{D}_k,p_k)$ is over proportionally sampled. In this case, the total number of test case n is updated proportionally to n_k (line 3), and for each sub-domain except the sub-domain $(\mathcal{D}_k,p_k)$, extra $(\lceil n.p_i \rceil - n_i)$ test case are selected (lines 4-6).

Otherwise, the sub-domains are under proportionally sampled, and for each sub-domain $(\mathcal{D}_i,p_i)$, extra $(\lceil n.p_i \rceil - n_i)$ test case are selected (lines 8-9).

D. Stopping Criteria

We define a test stopping criteria based on the tester required (i) maximal error of the reliability estimate d, and (ii) confidence level $(1-\alpha)$. The goal of reliability testing is then to estimate the reliability $\hat{R}$ to within d with $100(1-\alpha)\%$ probability.

The total required number of test cases depends on the allocation of the selected test cases to the sub-domains. Let a_l (as defined in Section II-D) be the allocation ratio for the sub-domain $\mathcal{D}_l$, with $n_l = n.a_l$. Also, let z be the upper $\alpha/2$ critical point of the standard normal distribution. Then, we want to find n such that $z[\text{var}[\widehat{R}]]^{1/2} = d$ (margin of error equation), where $\text{var}[\widehat{R}] = \sum_{i=1}^{L} {p_i}^2 \frac{{\sigma_i}^2}{n_i} = \frac{1}{n} . \sum_{i=1}^{L} {p_i}^2 \frac{{\sigma_i}^2}{a_l}$.

Solving the margin of error equation for n leads to: $n = \frac{z^2}{d^2} \sum_{i=1}^{L} {p_i}^2 \frac{{\sigma_i}^2}{a_l}$.

From (5), we can compute the total cost c^* required to achieve the desired level of accuracy as follows [7]:

$$a_l = \frac{p_l.\sigma_l/\sqrt{c_l}}{\sum_{i=1}^{L} p_i.\sigma_i.\sqrt{c_i}} \text{ and } c^* = \frac{z^2}{d^2} \left[\sum_{i=1}^{L} p_i.\sigma_i.\sqrt{c_i} \right]^2 . \quad (7)$$

From here, we can compute $n_l = c^*.a_l$, and then, ultimately n.

E. Adaptive Constrained Test Selection Algorithm

Based on the discussions above, the adaptive constrained test selection algorithm works as described in Algorithm 2. In the intialization phase of the algorithm (lines 6-7), $|\mathcal{T}_{(\mathcal{D}_i,p_i)}|$ test case are selected from each sub-domain $(\mathcal{D}_i,p_i)$ based on a given initial number of test case n_{start}. $\mathcal{T}_{(\mathcal{D}_i,p_i)}$ represents the set of test cases selected from sub-domain $(\mathcal{D}_i,p_i)$. In the sampling phase (lines 10-27), the algorithm computes for each sub-domain the optimal required number of test cases to be select based on the stopping criteria formula in equation 7 (line 12-13). Extra test cases are then selected if required (lines 15-16). Otherwise, test cases have been optimally selected from that sub-domain (line 18). The algorithm computes the variance of the observed failure rate for each sub-domain after each sampling phase (line 24), and adjust the test allocation toward 100% similarity to the OP. The algorithm stops and returns the estimated reliability if (i) a maximal allowed test time interval Δ has passed or (ii) for all sub-domains the optimal required number of test cases has been selected and the total selected test cases are 100% similar to the OP (line 21).

 ISBN: 978-1-61208-367-4

Require: $OP = \{(\mathcal{D}_i, p_i) | i \in \{1, ..., L\}, \sum_{i=1}^{L} p_i = 1\}$
$\mathcal{T}_{OP} = \{\mathcal{T}_{(\mathcal{D}_i,p_i)} | (\mathcal{D}_i, p_i) \in OP\}$
Δ : maximal allowed test time
n_{start} : initial number of test cases to start
$1 - \alpha$: confidence level
d : margin of error
for $(\mathcal{D}_i, p_i) \in OP$ **do** // 1. Initialization
$|\mathcal{T}_{(\mathcal{D}_i,p_i)}| \leftarrow \lceil n_{\text{start}} . p_i \rceil$

end for
//2. Adaptive optimal constrained stratification
while true **do**
for $(\mathcal{D}_i, p_i) \in OP$ **do**
$c^* = \frac{z^2}{d^2} \left[\sum_{i=1}^{L} p_i . \sigma_i . \sqrt{(1 - p_i)} \right]^2$
$a_i = \frac{p_i . \sigma_i / \sqrt{(1-p_i)}}{\sum_{k=1}^{L} p_k . \sigma_k . \sqrt{(1-p_k)}}$
$n_i^o = \lceil c^* a_i \rceil$
if $|\mathcal{T}_{(\mathcal{D}_i,p_i)}| < n_i^o$ **then**
//select extra $(n_i^o - |\mathcal{T}_{(\mathcal{D}_i,p_i)}|)$ test cases from $(\mathcal{D}_i, p_i)$
$|\mathcal{T}_{(\mathcal{D}_i,p_i)}| \leftarrow n_i^o$
else
opt = opt + 1
end if
end for
if Δ passed or $(\text{opt} = L \wedge \mathcal{SC}(\mathcal{T}_{OP}) = 100\%)$ **then**
break;
end if
update statistics for all $(\mathcal{D}_i, p_i)$
Adjust to proportional sampling: call Algorithm 1
opt = 1
end while
return $\widehat{R} = \sum_{i=1}^{L} p_i . \widehat{R}_i$

Figure 2. Adaptive constrained test cases selection

IV. Experimental Evaluation

We conduct a set of experiments on a real subject program to evaluate the performance of the Adaptive Constrained Test Selection (ACTS) algorithm against the standard proportional test selection approach as proposed by Musa [6] (PS), and the theoretical optimal test selection approach (OS) with respect to the estimated reliability accuracy and precision. For (OS), we assume that we know the failure rates in advance, and we sample accordingly.

A. Experiment Design and Setup

1) Subject Program and Operational Profiles: Space: a language-oriented user interface developed by the European Space Agency. It allows the user to describe the configuration of an array of antennas with a high level language. The correct version as well as the 38 faulty versions and a test suite of $13,585$ test cases are downloaded from the software-artifact infrastructure repository [8]. In these experiments, three faulty versions are not used because we did not find test cases that failed on these faulty versions. Space is 9126 LOCs big.

A failure of an execution is determined by comparing the outputs of the faulty version and the correct version of the program. A failure is a deviation from the expected output. The failure rates for both studied programs are empirically computed by executing all the available test cases against each faulty version of a program and recording the number of failed test cases.

Operational profiles for Space are not available. We create operational profiles for Space as follows. We assume that in each sub-domain $\mathcal{D}_i$, all possible inputs are equally likely to arise. Hence, it follows that the number of sub-domains (greater or equal to two sub-domains) as well as the number of inputs in each sub-domain may not bias the statistical properties (i.e., variance and mean) of the estimated reliability. The estimated reliability is influenced by the probability of occurrence of the sub-domains, as well as the true failure rate of the tested software when executed with inputs from each sub-domain. In that sense, we partition the test cases of Space in six disjoint sub-domains. All six sub-domains contain the same number of test cases except for rounding issues. For each sub-domain, test cases are randomly selected without replacement from the pool of test cases. The $13,585$ test cases of Space are partitioned into six disjoint classes: 2264, 2264, 2264, 2264, 2264 and 2265. In order to minimize possible bias due to the choice of the test cases in each sub-domain, we repeat the allocation of the test cases of each subject program into the six sub-domains twice. This results into 2 possible allocations of the test cases to sub-domains $\mathcal{D}_i$ for each subject program.

Due to time and space limitation, not all possible operational profiles can be adopted in the experiments. We define two different profiles for the probability of occurrence of the sub-domains: (i) uniform profile: the probability of occurrence of each sub-domain is the same except for rounding error and (ii) optimal profile: the probability of occurrence of each sub-domain is proportional to the number of test cases allocated to each sub-domain using optimal allocation

These two profiles are some typical or extreme profiles and cannot represent all usage scenarios in field use. Consequently, for each subject program, 4 different operational profiles are created.

2) Performance Metrics: ACTS, PS and OS are randomized test selection strategies. For statistical significance, we conduct 200 independent repetitions of each experiment for each test selection strategy.

We compare the performances of ACTS, PS and OS by comparing the accuracy and precision of the estimated reliability by each approach. The accuracy of an estimate is a measure of how close the estimated value is to its true value. The precision of an estimate is a measure of how close the estimates measured from different samples are to another, when the samples are taken from the same data set. We use the sample variance as metric for the reliability estimation accuracy. The sample variance is an unbiased estimator of the variance. We use the *root mean squared error* (RMSE) to quantify the estimate precision.

Based on assumption 5 in Section II-E, the reliability estimates delivered by ACTS, PS, and OS are unbiased. Consequently, we can compare the relative efficiency of the estimates using the sample variance. For each experiment $\mathcal{E}$ we define the mean value of the reliability estimate ($\overline{R}$), its sample variance ($S^2_{199}(\widehat{R})$), its root mean squared error ($RMSE(\widehat{R})$), and the relative efficiency of the reliability estimator using ACTS to PS and OS as follows:

 ISBN: 978-1-61208-367-4

$$\overline{R} = \frac{1}{200}\sum_{i=1}^{200}\widehat{R_i},\quad S^2_{199}(\hat{R}) = \frac{1}{199}\sum_{i=1}^{200}(\widehat{R_i}-\overline{R})^2,\quad RMSE(\hat{R}) = \sqrt{\frac{1}{200}\sum_{i=1}^{200}(\widehat{R_i}-R)^2}$$

$$\text{eff}(\widehat{R}_{ACTS},\widehat{R}_{PS}) = \frac{RMSE(\widehat{R}_{PS})}{RMSE(\widehat{R}_{ACTS})},\quad \text{eff}(\widehat{R}_{ACTS},\widehat{R}_{OS}) = \frac{RMSE(\widehat{R}_{OS})}{RMSE(\widehat{R}_{ACTS})}$$

where R is the true reliability calculated based on the true failure rates, $\widehat{R_i}$ the reliability estimate in repetition i of the experiment, $\widehat{R}_{ACTS}$ the reliability estimate using ACTS, $\widehat{R}_{PS}$ the reliability estimate using PS and $\widehat{R}_{OS}$ the reliability estimate using OS.

The differences in reliability mean values between the different test selection strategies is confirmed using the non-parametric Mann-Whitney U test [9]. The differences between the sample variances are tested using the Brown-Forsythe test [9].

For each experiment and for each test selection strategy, we compute the reliability estimate at seven checkpoints: $200, 250, 350, \ldots, 500$. After 200 repetitions of the experiment, we compute the mean value, sample variance and the root mean square error of the reliability estimates for each test selection strategy. Note that the more test cases are executed the more will the variance of the estimator decrease. In addition, the experimental dataset is selected randomly from the population and the selection is repeated 200 times. Consequently, the selected dataset do not affect the efficiency and the generalizability of ACTS.

B. Experimental Results

The goal of this set of experiments is to assess the efficiency and precision of our reliability estimation approach.

Figure 3 presents the sample means and sample variances for Space. The dashed lines are the true reliability values for the subject programs.

According to the experimental results, the means as well as the sample variances of the reliability estimates of ACTS are closer to the true values than those of PS and OS. This is confirmed by the statistical tests Mann-Whitney U test and Brown-Forsythe test in table I. The table confirms that ACTS significantly deliver more accurate reliability estimate that PS and OS.

The computed mean of the relative efficiency of the reliability estimator using ACTS compared to the one using PS for the Space experiments was $1,57$. This means that PS will yield a reliability estimate as accurate as ACTS only if 57% more test cases are selected.

The computed mean of the relative efficiency of the reliability estimator using ACTS compared to the one using OS for the Space experiments was $1,32$. This means that OS will yield a reliability estimate as accurate as ACTS only if 32% more test cases are selected.

V. Related Work

Stratified sampling is linked to the idea of partition testing or sub-domain testing of a software. Techniques to estimate software reliability using partition testing are proposed by Brown and Lipow [10] and Duramn and Wiorkowski [11], for example. They introduced the idea of sampling to reliability estimation but did not specify a sampling design. Podgurski et al.'s [12] work of is the most related work to our research. However, they only used the idea of equal stratification using clustering to estimate the software reliability from software execution profiles collected by capture/replay tools. Failure rates have been extensively used in the area of adaptive random testing (for example Cangussu et al.'s [13] and Chen et al.'s [14]). Adaptive random testing aims to distribute the selected test cases as spaced out as possible to increase the chance of hitting the failure patterns. The intuition behind adaptive random sampling can be added in a future work to our approach to probably further enhance the efficiency of the reliability estimator. Cangussu et al.'s [13] and Chen et al.'s [14] do not address the problem of reliability estimator efficiency.

TABLE I. Mann-Whitney U and Brown-Forsythe test results for the sample means and variances for Space

Scenarios		Variance ACTS	Variance OS	Mean ACTS	Mean OS
Space pro-file1	PS	0/7	4/7	6/7	0/7
	OS	0/7	-	7/7	-
Space pro-file2	PS	0/7	1/7	7/7	7/7
	OS	1/7	-	7/7	-
Space pro-file3	PS	1/7	1/7	7/7	5/7
	OS	1/7	-	5/7	-
Space pro-file4	PS	0/7	1/7	5/7	1/7
	OS	2/7	-	6/7	-

Thévenod-Fosse and Waeselynck [15] present the usage of probabilistic test generation for fault detection. They generate automatically tests to address different behavioral and structural test criteria. Apparently, Thévenod-Fosse and Waeselynck [15] view the evaluation of tests as *inexpensive*. They call their approach "statistical testing" although it dos not involve reliability estimation. In contrast to Thévenod-Fosse and Waeselynck [15], we think that evaluating test is an expensive process. Our approach aims to reduce the variance of a reliability estimator and consequently reduce the required number of executed and evaluated test cases to reach a target reliability confidence. A recent work on adaptive testing by Junpeng and Cai [16], allocates test cases using a gradient search method based on the variance variation of the failure rate. However, their approach introduces bias resulting from the use of the gradient method: it is possible that all test cases are selected from the sub-domain that first reveals a failure. They avoid such situations by introducing a biased estimator using Bayesian estimation.

VI. Conclusions and Future Work

Statistical testing is in the most of the cases impractical due to the large number of test cases required to reach a target reliability. In this paper, we presented an approach to automatically select test cases from an operational profile sub-domains with the goal to reduce the variance of the reliability estimator. Our initial experimental results are promising and shows that our approach ACTS outperforms PS and OS.

We plan to conduct further experiments to validate the

 ISBN: 978-1-61208-367-4

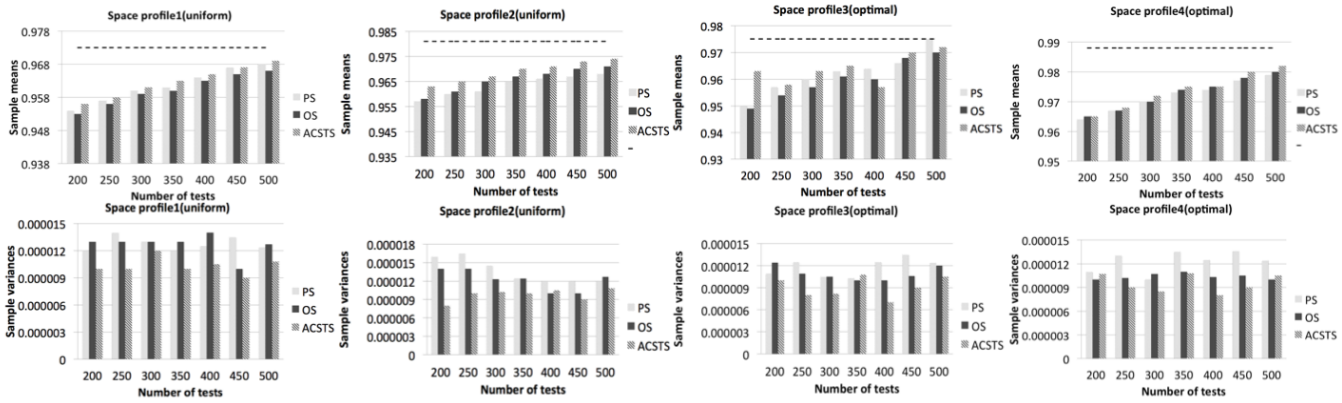


Figure 3. Sample means and sample variances of the reliability estimates for Space

effectiveness of ACTS on software with real specified operational profiles. We also plan to further investigate the efficiency of our approach for ultra-high software reliability scenarios.

ACKNOWLEDGMENT

This work was partially supported by the German Federal Ministry of Economics and Energy (BMWI), grant No. 01MD11005 (PeerEnergyCloud).

REFERENCES

[1] M.-H. Chen, M. Lyu, and W. Wong, "Effect of code coverage on software reliability measurement," Reliability, IEEE Transactions on, vol. 50, no. 2, 2001, pp. 165–170.

[2] P. K. Kapur, D. N. Goswami, and A. Bardhan, "A general software reliability growth model with testing effort dependent learning process," Int. J. Model. Simul., vol. 27, no. 4, Sep. 2007, pp. 340–346.

[3] C. V. Ramamoorthy and F. B. Bastani, "Software reliability status and perspectives," IEEE Trans. Softw. Eng., vol. 8, no. 4, Jul. 1982, pp. 354–371.

[4] T. A. Thayer, M. Lipow, and E. C. Nelson, Software reliability : a study of large project reality, ser. TRW Series of software technology; 2. Amsterdam: North-Holland, 1978.

[5] R. W. Butler and G. B. Finelli, "The infeasibility of quantifying the reliability of life-critical real-time software," IEEE Trans. Softw. Eng., vol. 19, no. 1, Jan. 1993, pp. 3–12.

[6] J. D. Musa, "Operational profiles in software-reliability engineering," IEEE Softw., vol. 10, no. 2, Mar. 1993, pp. 14–32.

[7] W. Cochran, Sampling techniques, ser. Wiley series in probability and mathematical statistics: Applied probability and statistics. Wiley, 1977.

[8] "The software-artifact infrastructure repository," http://sir.unl.edu, accessed: 2014-08-30.

[9] S. Wilks, Mathematical Statistics. Read Books, 2008.

[10] J. R. Brown and M. Lipow, "Testing for software reliability," SIGPLAN Not., vol. 10, no. 6, Apr. 1975, pp. 518–527.

[11] J. W. Duran and J. J. Wiorkowski, "Quantifying software validity by sampling," Reliability, IEEE Transactions on, vol. R-29, no. 2, June 1980, pp. 141–144.

[12] A. Podgurski, W. Masri, Y. McCleese, F. G. Wolff, and C. Yang, "Estimation of software reliability by stratified sampling," 1999.

[13] J. W. Cangussu, K. Cooper, and W. E. Wong, "A segment based approach for the reduction of the number of test cases for performance evaluation of components," International Journal of Software Engineering and Knowledge Engineering, vol. 19, no. 04, 2009, pp. 481–505.

[14] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. H. Tse, "Adaptive random testing: The art of test case diversity," J. Syst. Softw., vol. 83, no. 1, Jan. 2010, pp. 60–66.

[15] P. Thévenod-Fosse and H. Waeselynck, "Statemate applied to statistical software testing," in Proceedings of the 1993 ACM SIGSOFT International Symposium on Software Testing and Analysis, ser. ISSTA '93. New York, NY, USA: ACM, 1993, pp. 99–109.

[16] B.-B. y. Junpeng Lv and K. yuan Cai, "On the asymptotic behavior of adaptive testing strategy for software reliability assessment," Transaction on software Engineering, vol. 40, no. 4, April 2014, pp. 396–412.

 ISBN: 978-1-61208-367-4

PRReSE – Process of Non-Functional Requirements Reuse for Embedded Systems Based on a NFR-Framework

Cristiano Marçal Toniolo
Faculty of Exact and Natural Sciences
Methodist University of Piracicaba (UNIMEP)
Piracicaba – São Paulo – Brazil
e-mail: cmtoniolo@gmail.com

Luiz Eduardo Galvão Martins
Institute of Science and Technology
Federal University of São Paulo (UNIFESP)
São José dos Campos – São Paulo – Brazil
e-mail: legmartins@unifesp.br

***Abstract* — The Embedded Systems are increasingly present in society's daily life. Their demand in several home appliances makes them more complex, bringing the necessity of a more careful requirements engineering than for traditional systems. The requirements reuse for embedded systems, especially those addressed to non-functional requirements, is still a challenge for industries that develop products based on Embedded Systems (ES). This paper presents a Process of Non-Functional Requirements Reuse for Embedded Systems – called PRReSE - using NFR-Framework as an approach to improve the concept, design and development of such systems. The process was instantiated in a case study to illustrate the reuse of non-functional requirements in a product family; the family chosen was for microwave oven.**

Keywords-Embedded Systems; Requirements Reuse; Non-Functional Requirements; NFR-Framework.

I. INTRODUCTION

Software Engineering and Requirements Engineering work together to find new ways to ensure the quality of software development. To achieve this, a step of great importance in the process is the requirements elicitation, which seeks the understanding of the user's needs. The elicitation process defines and documents the steps so that an organization can elicit, analyze, specify and verify the requirements [15].

The advancement of techniques and methodologies allows us thinking about a systematic requirements reuse throughout the project development. According to Sommerville and Sawyer, the requirements reuse saves time and efforts in their elicitation. About 80% of the requirements are reused when dealing with similar systems [10].

The requirements reuse is performed in several ways, e.g., software components and requirements to make the reuse process even more efficient and able to answer the market demands. However, the reuse performed in industries is somehow intuitive, since engineers and developers reuse methods and documents in new projects based on their own experience.

Non-functional requirement is a central concept in this work, which means a quality feature that can affect the entire system to be developed. This study presents a Process of Non-Functional Requirements reuse for Embedded Systems (PRReSE is a Portuguese acronym for *Processo de Reuso de Requisitos Não-Funcionais para Sistemas Embarcados*). The proposed process is based on a NFR-Framework [16], which is a method to assist engineers and software designers to produce software in a faster and with more quality way, in a high level of quality with the lowest possible cost. This is precisely the role of engineering, namely, look for best quality systems within a cost compatible with this quality [8].

In industry context, many projects are related to each other and their requirements can be stored and reused in new projects in the future. Such projects can be divided into innovation of previous projects or into product families.

The innovations are related to implementations of new features in products that do not have them yet. So, a new version can be available. Product Families are related to new versions of products from the same family, e.g., microwave ovens – or to the creation of new ones, though with features previously used.

Therefore, the motivations for this study is the fact that the requirements reuse is a widely used approach to management, web, financial and administration systems, but poorly used in embedded systems and even more if dealing with non-functional requirements.

This paper is organized as follows: Section 2 presents background and related works in requirements reuse. Section 3 shows PRReSE process. Section 4 presents the results and analysis of the performed case study. Section 5 concludes the paper and points out to future works.

II. REQUIREMENTS REUSE

Software Requirements have to be carefully elicited in order to not compromise the whole systems development. As discussed in Kotonya and Sommerville "requirements are defined at the first phases of the system development as being a specification to be implemented" [2].

Requirements describe the user's needs guiding developers how the system must behave, where it has to be applied and with some quality constraints. Several techniques have been used in order to solve the problems of reuse. One of them is the requirements reuse which, according to Sommerville and Sawyer, occurs "when developing requirements for a new system is necessary, wherever possible, reuse requirements from other systems which have been developed to the same area of application",

 ISBN: 978-1-61208-367-4

i.e., the same field of the system [10]. Reuse can reduce the cost, coding and testing of the project if it is systematically done; therefore, reducing the effort of new elicitation for several applications [2][3][9].

The advantages to adopt requirements reuse are the elicitation time saving, analysis and requirements validation, reduction of the risk of new elicitations that might hinder the requirements implementations, leading to a requirements reuse without alterations, or with minimal settings leaving the elicitation process only for new requirements of the system. This leads the system's development life cycle to start earlier.

The identification, capture and organization of a requirements process with the purpose of reuse in new systems can be considered a domain reuse approach. Kotonya and Sommerville show some situations where the reuse is possible [2]:

1. If the requirement shows information about the application domain: several requirements do not specify the system's functionality but presents its constraints or operation derived from the application domain.
2. If the requirement is consistent with the presentation of information style: if possible, common sense to organize, to have a consistent interface for all systems. It means that the user's errors are smaller when they change from one system to another.
3. When the requirements reflect the company's policies such as security and performance, they must be reflected in the system requirements. In this context, requirements are developed for the system and can be considered encapsulated requirements, which are common to a large number of different systems.

This way, for many systems, 50% of requirements are in such classes, which are a considerable scope for the cost reduction by requirements reuse [2]. Other reasons to perform the requirements reuse can be: requirements already analyzed tend to suffer few or no alterations; cost reduction of new requirements elicitation, which may lead to an incompatibility with other systems generating unexpected problems. The requirements reuse process has to be agreed for engineers and developers aiming to improve the system development cycle [13].

Some reuse processes can be cited as follows: analysis of domain, textual analysis, use cases patterns, scenarios, frameworks, direct and indirect reuse. The indirect reuse is specified as follows [10]:

1. Identify the requirements that are close or similar to the stakeholders requirements for the system being developed.
2. Show these requirements to the stakeholders and explain their meanings.
3. Ask where the requirements would be adequate or inadequate.
4. Rewrite the requirements according to suggestions and repeat the process until all the stakeholders agree with them.

The elicitation steps for direct reuse are as following:

1. Identify the common requirements between the existing system and the one to be developed.
2. Recognize the potentially reusable and relevant requirements in the existing system to identify the common features.
3. Evaluate the possible reusable requirements with the purpose of validate them with the stakeholders for the new system to be developed.
4. Check with users if the requirements meet their needs.

A product family approach is another usual way of requirements reuse. This process is based on two concepts: strong reuse and weak reuse [7]. In the strong reuse, the requirements must be synchronized with the associated products, and any alterations on them affect the entire product family.

In the weak reuse, the requirements are copied from the beginning of the project and they can evolve from other requirements.

Another way to identify product families, according to Lam, McDermind and Vickers [4], is that "requirements are sensitive to the context and are specified to a set of problems", then in product families it is possible:

- To identify commonalities between the system "father" and the system "son";
- To impose a common requirements engineering process or a pattern inside the organization;
- To antecipate some types of alterations and specializations;
- To recognize labor patterns to assist the planning of the project [4].

The reuse problems may go through some issues that make them a difficult task, such as: the engineering methods and techniques that are not specifically designed to reuse; the process which does not prioritize an integrated development and exchanging experience among the team members; the organization which works with individual projects without reuse planning; and the business that aims profit and works with financial return only.

Sommerville [11] lists a number of problems such as: maintenance cost, lack of tools and specific development, "non-invented-here" syndrome, creation of a library or repository to store the components. These factors impact the development costs. When it comes to reuse, it is possible to have models for requirements patterns, reuse of documents and artifacts, which makes this area very embracing and without a definite pattern to reuse system requirements along with the lack of specific tools but with several techniques addressed to each type of context and problem.

III. PRReSE: Process of Non-Functional Requirements Reuse for Embedded Systems Based on a NFR-Framework

Studies are performed in order to shorten the steps of the development process and consequently save time with development and cost reduction. Researches has shown that many efforts have been made to reuse software components and requirements, aiming to accelerate the development process in computer programming since the market competition is increasingly fierce. The paper focuses on a Process of Non-Functional Requirements Reuse for

 ISBN: 978-1-61208-367-4

Embedded Systems based on a NFR-Framework, which intends to become a guidance for professionals in appliance industry, assisting them when developing new systems reusing non-functional requirements for embedded system previously developed.

A. PRReSE: General Flow

PRReSE starts with the phase of separation of artifacts which can generate inputs for the requirements reuse activities [9][12]. These inputs correspond to previously analyzed data, which are considered as requirements feeding the requirements reuse process. According to [5][6][9], the following documents can be considered as being input artifacts: questionnaires, reference models, checklists, documents based on patterns, documented interviews, catalogues or technical descriptions of a system. Other artifacts can be found in *IEEE Std 830-1998* models and in *Volere Template* [1][14]. The general model for the requirements process described in this study is shown in Figure 1. The main activities of the proposed process are explained in the next sections.

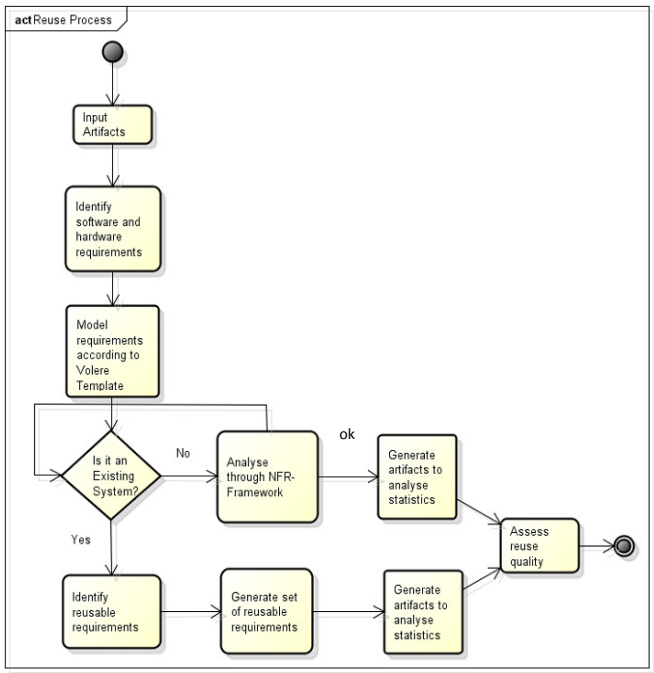


Figure 1. General Flow of Requirements Reuse Process for Embbeded Systems (PRReSE).

B. Identification of Software and Hardware Requirements

In this phase, the requirements engineer has to identify, select and prioritize requirements through input artifacts with the purpose of producing output artifacts to the next step, which can be divided into software and hardware non-functional requirements.

Input artifacts are the analyzed data in the documents collect from the stakeholders, whose specific goal is to create knowledge of the software context to be modeled by the requirements engineer, electronic engineers and technicians. Technical manuals and catalogues can be cited as examples of such documents. After separating these documents, requirements engineering process is in charge to identify the non-functional requirements for software and hardware. The next step is modeling according to Volere Template.

C. Modeling Non-Functional Requirements according to Volere Template

After the first phase of PRReSE, when the requirements document is generate, the next step is to create the non-functional requirements cards and store them to be used in NFR-Framework.

The non-functional requirements documents created in the previous step are the start up for the modeling process. Thus, the non-functional requirements can be catalogued using the *Volere Template* cards. The output artifact for this PRReSE step is the creation of all non-functional requirements cards to model according to NFR-Framework.

D. Checking the Existence of Legacy System

This step shows two conditions when modeling an embedded system: the existence of a legacy system or the lack of a legacy system.

E. Lack of a Legacy System

An analysis to verify the existence of a modeled and compatible system is performed after to model non-functional requirements according to *Volere Template* to apply PRReSE. Otherwise, the following steps are:

1. When the system under consideration has not been elicited or do not have any relationship with another system – *product families* – it is necessary to perform all the analysis for the artifacts described by the NFR-Framework. This means creating a catalogue of knowledge to form a basis to research future systems.
2. Develop SIG graphs to model the relationship among non-functional requirements and expose them graphically.
3. Generate the modeled graphs for the entire systems, as output artifacts.

F. Existence of Legacy System

The analysis to verify the existence of an already modeled system compatible to apply PRReSE is performed after the creation of the *Volere Template* cards. The steps are:

1. Identify the reusable requirements according to the following procedures:
   a. If there is any relation to some existing product or product families, search the catalogues already created, verify the presence of requirements that can be reused and analyze only the compatible requirements.
   b. This identification will be performed through the comparison of the requirements modeled in SIG graphs with the requirements collected in the input artifacts (*Volere* cards). Thus, when the requirements are in the upper levels of the graph,

 ISBN: 978-1-61208-367-4

the reuse will have little or no alterations at all. In the lower levels of the graph, the reuse will tend to be performed.

2. Generate a set of reusable requirements: it happens after the comparison with the legacy systems. Thus, the SIG graphs are created and the candidate requirements for reuse are identified.
   a. Identical requirements are reused without any alteration.
   b. Requirements with some similarities to a specified NFR must be reused, with the necessary alterations. Also, they must be identified in the SIG graphs with dashed circle.
   c. Requirements that are not catalogued must be entirely elicited and represented in the SIG graphs by a solid circle.
3. Create Artifacts for Statistical Analysis: after creating SIG graphs and identifying the requirements which were or were not entirely used, they are quantified and tagged as entirely used, partially used or not used. Then the steps bellow are followed:
   a. Recover all the stored SIG Graphs;
   b. Quantify the requirements entirely used;
   c. Quantify the requirements partially used;
   d. Quantify the requirements not used.

G. Analysis of Reuse Quality

This analysis defines the reuse viability and how the indicators quantify the reuse of non-functional requirements. From these results, the quantity is verified and a reuse pattern of quality will be established in percentage:

a. Reused without alterations;
b. Reused with alterations;
c. New requirements elicited with the stakeholders.

Such analysis must be performed based on the equation (1) and Table 1:

$$(1) \quad RP = (QRR / QR) * 100$$

where:
RP = Reuse Percentage
QRR = Quantity of reused requirements
QR = Total quantity of requirements

TABLE I. REUSE INDEX FOR NON-FUNCTIONAL REQUIREMENTS

Reuse Index	Reuse Percentage
High	RP ≥ 85%
Adequate	RP ≥ 60% e RP < 85%
Insufficient	RP ≥ 40% e RP < 60%
Inadequate	RP < 40%

Table 1 presents indicators which may assist software engineers to get an idea of how much will be necessary to elicit new software from existing requirements and the quantity of reuse it may be generated providing time and profit.

H. Analysis of the Obtained Results

This analysis defines which percentage of requirements reuse was obtained with PRReSE, and also if the work of preparing the software design will become viable, consuming less effort from those involved in the project. It will also allow the software and requirements engineers to get the parameters to develop new projects, such as time to analyze and elicit requirements, making this process to work as a knowledge and learning basis with experiments from previous projects.

IV. CASE STUDY: MICROWAVE OVEN

This case study performed the steps of the Reuse of Non-Functional Requirements Process for Embedded Systems showing that PRReSe becomes a feasible alternative for requirements reuse for embedded systems.

A. Requirements Identification

The requirements for the microwave oven family were extracted from the catalogue of products previously studied, because there is a lack of documentation related to the embedded systems used in microwave ovens. This way it was possible to elaborate the requirement cards according to the model suggested by *Volere Template*.

Panasonic's microwave oven manual of the flat and family models were used as input artifacts, and the following non-functional requirements were extracted: security, usability, customization, learning, accessibility and capacity.

B. Non-Functional Requirements Cards

The requirements cards based on *Volere Template* were fulfilled after the requirements identification. Such cards offer a pattern structure to describe the requirements, which turn easy the work of requirement engineers during the reuse process.

C. Non-Functional Requirements Catalogue

Non-Functional Requirements Catalogues are modeled for embedded systems (ES) after the requirements and the cards get ready. These catalogues have the purpose to show non-functional requirements in a hierarchical form with the generic NFRs displayed above the more specific ones, as showed in Figure 2.

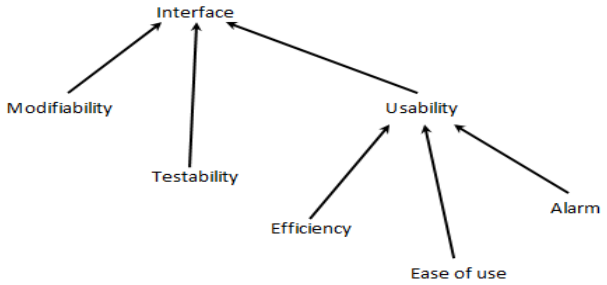


Figure 2. Interface Non-Functional Requirements for Microwave Oven.

The SIG graphs for such requirements were subsequently modeled, and the resulted specification model are presented in Figure 3. The reuse model illustrated in the figure shows three levels of requirements, which are: requirements without alterations - cloned objects, requirements with some type of alterations – derived objects, and new requirements specified for the project being developed – new objects. The

requirements for the Panasonic Flat-Style microwave NF-SF560WRU™ were initially modeled becoming the base for the requirements related to the product family, as shown in Figure 4. This model of oven has simple features and basic commands. Monzon [7] explains that to have the requirements reuse done there must be a set of common requirements, which will be reused in new projects. This set of requirements becomes the core for the reuse in embedded systems projects or a product family, which will be reused in each evolution or new version of a product.

Figure 3. Model of Non-Functional Requirements Reuse from a common requirements set [7].

D. Systems Evolution

The model NN-GF580MRU™ from the Panasonic Flat Family™ was used and the requirements were elicited and analyzed. SIG graphs were created to compare it with the previous model – according to Figures 5 and 6. The softgoals which needed to be elicited from the catalogue of product due to advanced features were added to the core requirements already presented in the previous model. An analysis of a model of a microwave oven, which is not part of the adopted product family was performed to show the requirements reuse process extent. The used model was NNST669WRU™ with innovative features. The SIG graph referring to the model above was created to become possible the comparison with the core requirements in the previous models, as illustrated in Figure 6.

E. Analysis of the Results

Analysis of the results was performed from the creation of both requirements cards and SIG graphs, according to what has been proposed in the non-functional requirements process. Table 2 summarizes the requirements from the reuse process, which are commented in the next sections.

TABLE II. RESUME OF NON-FUNCTIONAL REQUIREMENTS OF MICROWAVE OVEN

Model	(a)	(b)	(c)	(d)	(e)	Reuse
NN-SF560WRU	31	0	0	0	0	0%
NN-GF580MRU	33	31	31	0	2	94%
NN-ST669WRU	35	31	31	1	3	88%

(a) Non-Functional Requirements Total
(b) Reused Requirements Total
(c) Reused Requirements without alterations
(d) Reused Requirements with alteration
(e) New Requirements

F. Reused requirements without alterations

The model NN-SF560WRU was the first to be analyzed and 31 non-functional requirements were elicited. The second model, NN-GF580MRU had 33 non-functional requirements elicited and 31 of them were reused from the first model, which correspond to 94% of requirements reuse. The third model NN-ST669WRU had 35 non-functional requirements elicited. From this total, 31 requirements were reused, meaning 88% of reuse.

G. Reused requirements with alterations

The first two models of microwave oven used in this case study belong to the same product family and the third model belongs to a different product family. The non-functional requirements "stand by key" in the model NN-GF580MRU was the one reused with alterations. Here, the "stand by key" was matched to the "clock key" resulting in a "stand by/clock key". This model had 35 elicited requirements, which 2.8% correspond to requirements reused with alterations.

H. New Requirements

Two new requirements were created for the second model, which belongs to the same family of the first model. These two new requirements mean 6% of the elicited requirements. For the third model – which does not belong to the family of the first one – three new requirements were created, meaning 8.5% of the elicited requirements.

I. Reuse Analysis

The Requirements Reuse in the second model of microwave oven analyzed saved time in the analysis of requirements phase since the biggest efforts were performed in the base model. Thus, the requirements reuse in products of the same family leaded to a significant time saving.

The study case showed that with PRReSE adoption was possible to obtain 94% of requirements reuse in the second model in relation to the first one, which belong to the same family indicating a promising reuse process.

PRReSE allowed 88% of requirements reuse in the third model of microwave oven in relation to the first, even considering that the third model belongs to a different product family. This result also can be considered as a promising one. Since this model belongs to a different family, could be expected a lower percentage of reuse.

V. CONCLUSION AND FUTURE WORK

The requirements reuse in traditional systems – not embedded - inspired PRReSE creation in a yet not enough explored context, i.e., non-functional requirements for embedded systems. NFR-Framework was adopted because it is a specific methodology to NFRs, and also it is largely known in the Requirements Engineering community.

The productivity of embedded systems development can significantly increase using the requirements reuse techniques, especially because in embedded systems development is very common to use product families. The statement above can be seen in the microwave oven case study, which applied PRReSE to reuse the requirements in three models, being two from the same family and one from a different product family.

There was a large effort to elicit requirements at the first model, then using PRReSE process it became easier to reuse

 ISBN: 978-1-61208-367-4

the resulting requirements in the following models. According to PRReSE, the visualization of non-functional requirements through SIG graphs becomes the reuse identification easier by the requirements engineering.

According to the evidences observed in the case study, during the creation of the second model 94% of non-functional requirements were reused from the first model – both belonging to the same product family. PRReSE allowed the reuse of 88% of the non-functional requirements in the third model, considering that this last one did not belong to the same product family.

As future work, a software tool to support PRReSE will be developed to facilitate its use. Another case study performing all steps of PRReSE is being planned; such study will be in the area of medical devices.

REFERENCES

[1] IEEE, "IEEE Recommended Practice for Sotware Requirements Specifications". IEEE Std 830-1998.

[2] G. Kotonya and I. Sommerville, Requirements Engineering: process and techniques, Ed. Wiley, 2001.

[3] W. Lam, "A case-study of Requirements Reuse through product families". In: Annals of Software Enginering, vol. 5, 1998, pp. 253 – 277.

[4] W. Lam, J. McDermind, and A. Vickers, "Ten Steps Towards Systematic Requirements Reuse", Proceedings of the Third IEEE International Symposium on Requirements Engineering. Jan/1999, pp. 6-15.

[5] X. Liu, S. Lui, and X. Zheng, "Adapting the NFR Framework to aspectual use-case driven approach", In: 7th ACIS International Conference on Software Engineering Research, Management and Applications. Dec/2009, pp. 209-214.

[6] O. López, M. A. Laguna, and F. J. Garcia, "Metamodeling for Requirements Reuse", In: Proceedings of the 5th. International Workshop on Requirements Engineering (WER'02). Valencia, Espanha Nov/2002.

[7] A. Monzon, "A pratical approach to Requirements Reuse in product families of on-board systems", In: 16th IEEE International Conference on Requirements Engineering. Sep/2008, pp. 223-228.

[8] A. R. C. Rocha, Qualidade de Software: teoria e prática. São Paulo : Prentice Hall, 2001.

[9] J. Rumbaugh, Object-Oriented Modeling and Design. International Publlisher: Prentice-Hall. International Pub., 1990.

[10] I. Sommerville and P. Sawyer, Requirements Engineering, John Wiley, 2000.

[11] I. Sommerville, Software Engineering, 9th Edition, Addison-Wesley, 2009.

[12] S. Supakkul, "Capturing, organizing, and reusing knowledge of NFRs: an NFR pattern approach", In: 2nd International Workshop on Managing Requirements Knowledge (MARK). Sep/2009, pp. 75-84.

[13] O. Villegas and M. A. Laguna, "Requirements Reuse for software development", In: 5th IEEE International Symposium on Requirements Engineering. Toronto, Canada, Aug/2001, pp. 27-31.

[14] J. Robertson and S. Robertson, Volere Requirements Specification Template. Atlantic Systems Guild London, 2011.

[15] K. L. Wiegers, "Requirements specification template". Microsoft Press, 1999.

[16] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, "Non-Functional Requirements in Software Engineering" In: The Kluwer International Series in Software Engineering, Vol. 5, 1999.

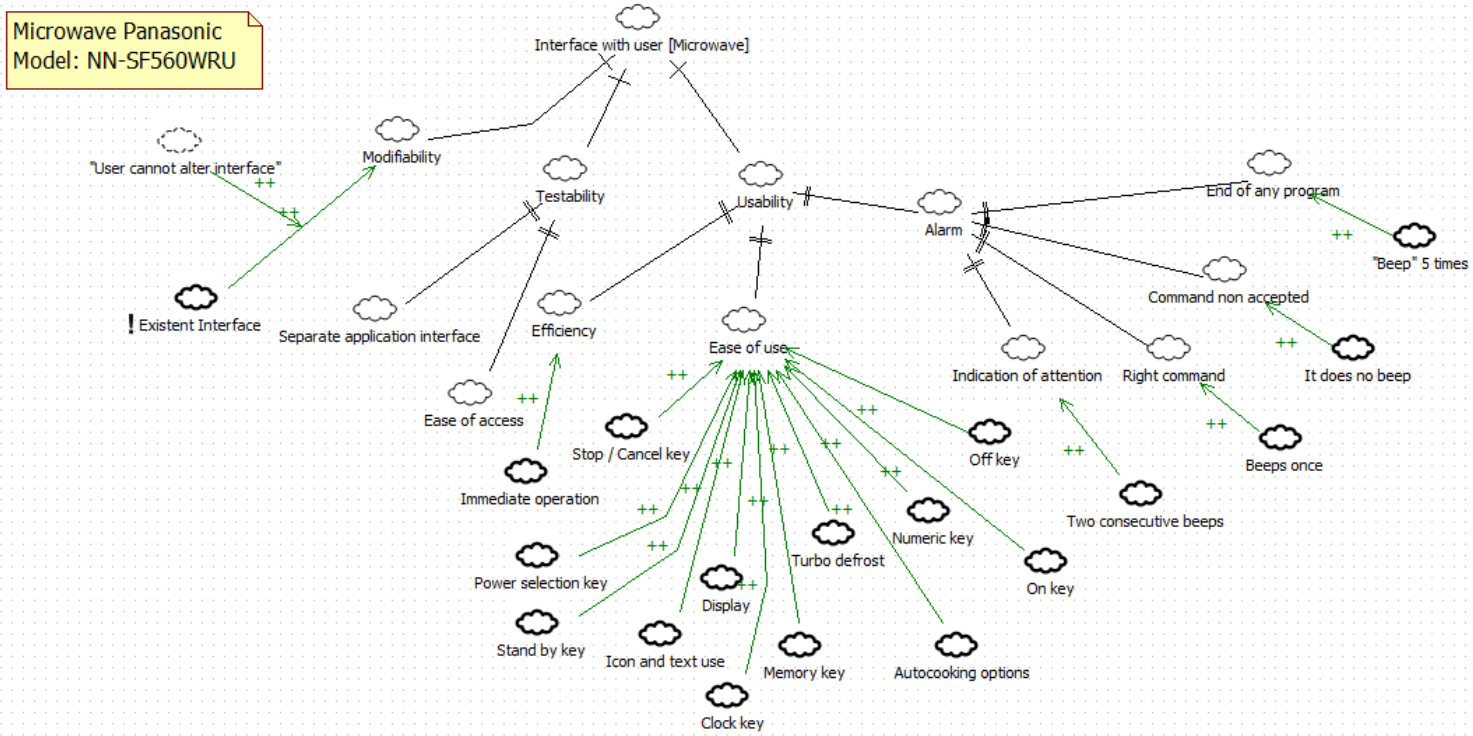


Figure 4. SIG Graph for "User Interface" requirement for microwave oven model NN-SF560WRU.

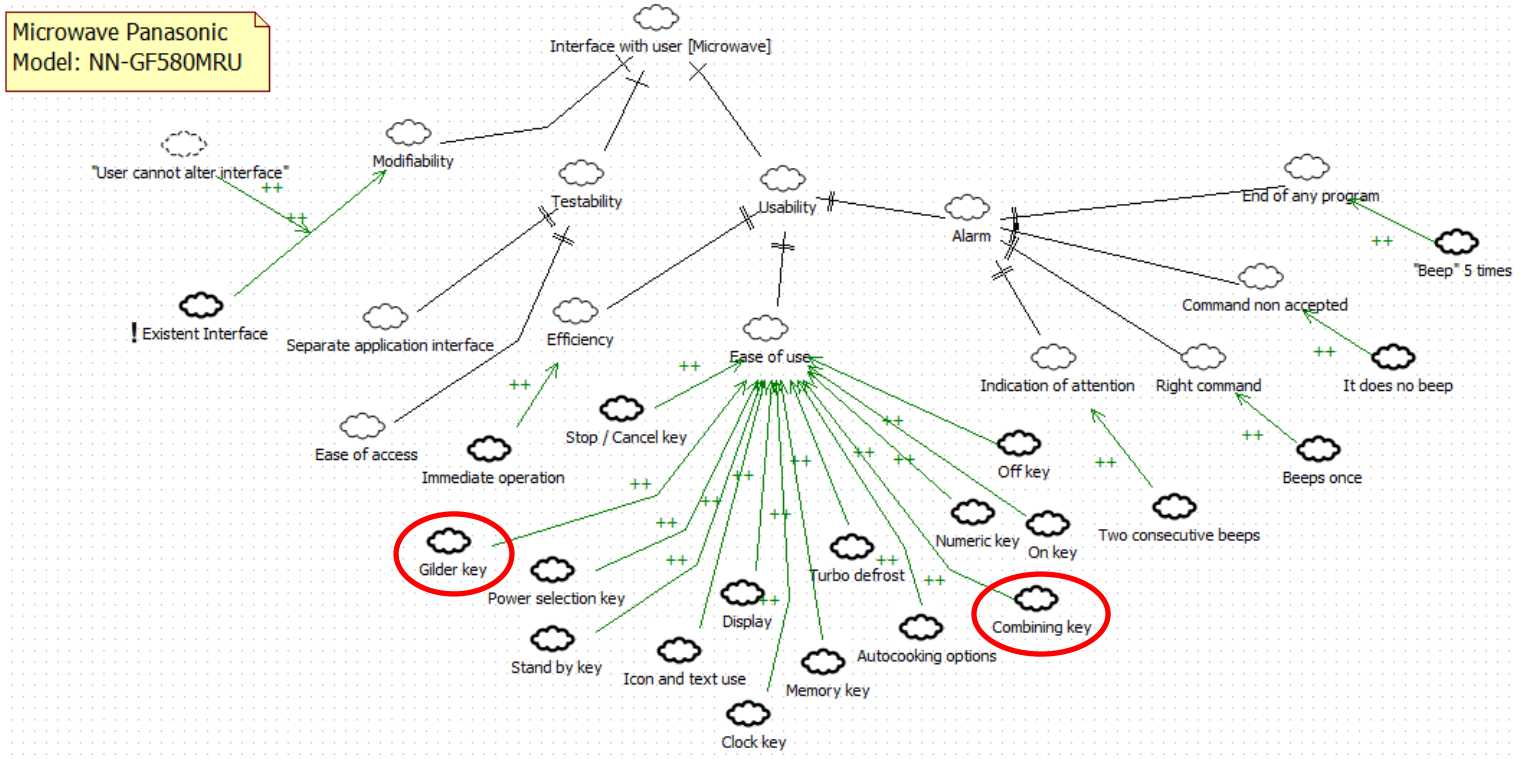


Figure 5. SIG Graph for "User Interface" requirement for microwave oven model NN-GF580MRU.

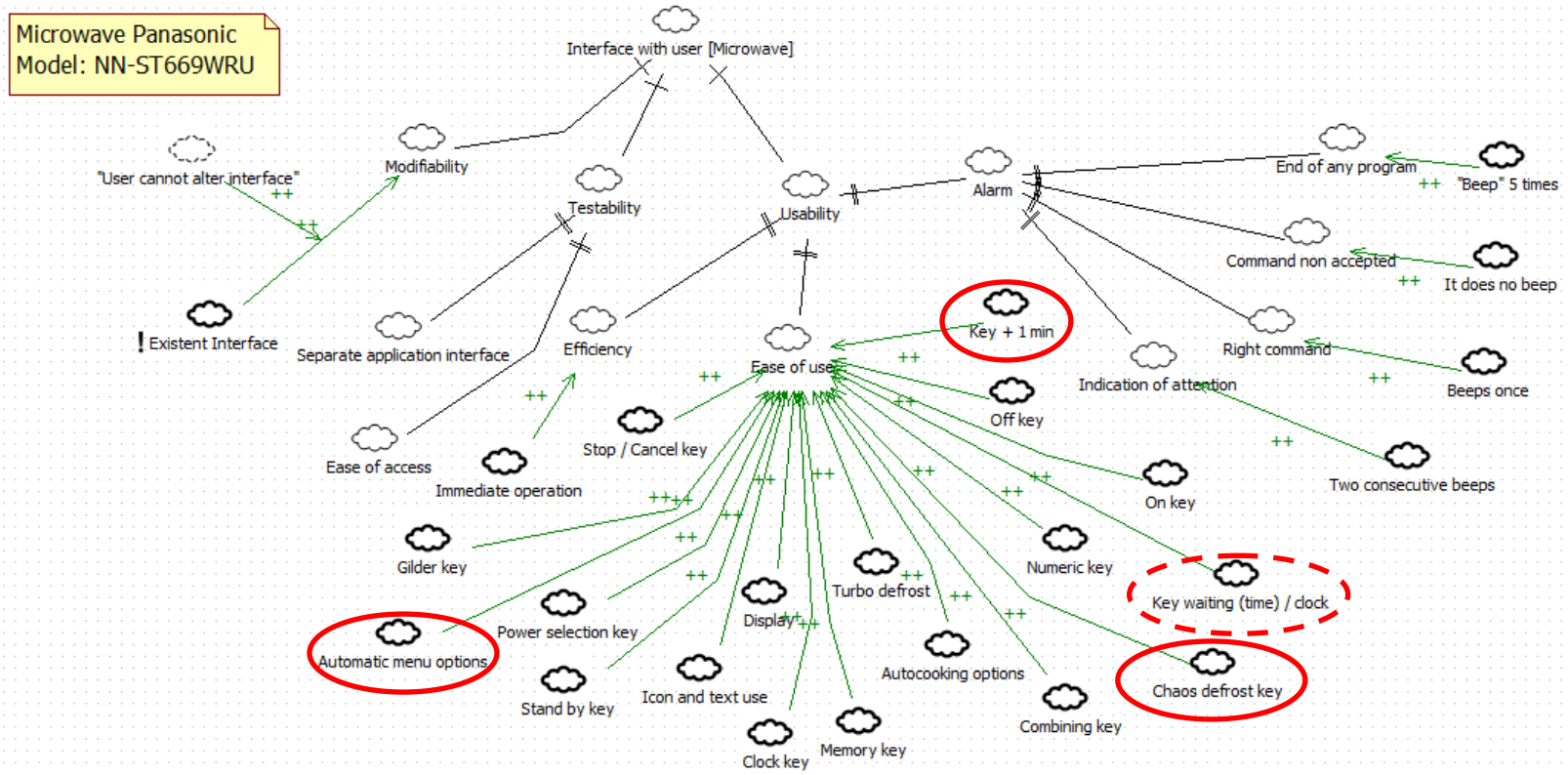


Figure 6. SIG Graph for "User Interface" requirement for microwave oven model NN-ST669WRU.

 ISBN: 978-1-61208-367-4

A Model-Driven Approach to the Development of Heterogeneous Software Product Lines

Thomas Buchmann and Felix Schwägerl
University of Bayreuth
Chair of Applied Computer Science I
Bayreuth, Germany
{thomas.buchmann, felix.schwaegerl}@uni-bayreuth.de

***Abstract*—Software product line engineering is dedicated to planned reuse of software components based upon a common platform, from which single products may be derived. The common platform consists of different types of artefacts like requirements, specifications, architecture definitions, source code, and so forth. Only recently, research projects have been started dealing with model-driven development of software product lines. So far, the resulting tools can only handle one type of artefact at the same time. In this paper, requirements, concepts and limitations of tool support for heterogeneous model-driven software product line engineering are discussed. As a proof of concept, an extension to the toolchain FAMILE is presented, which supports mapping of features to different types of artefacts in heterogeneous model-driven software projects at the same time.**

Keywords–software product lines; model-driven development; negative variability; feature models; heterogeneity.

I. INTRODUCTION

Software Product Line Engineering (SPLE) [1][2] deals with the systematic development of products belonging to a common system family. Rather than developing each instance of a product line from scratch, reusable software artefacts are created such that each product may be composed from a collection of reusable artefacts — the platform. Commonalities and differences among different products may be captured in a *feature model* [3], whereas *feature configurations* describe the characteristics of particular products by selecting or deselecting the respective features. Typical SPLE processes distinguish between *domain engineering*, which deals with the establishment of the platform as well as the feature model, and *application engineering*, which is concerned with the derivation of particular products out of the product line by exploiting and binding the variability provided by the platform.

Two distinct approaches exist to realize variability in SPLE: In approaches based upon *positive variability*, product-specific artefacts are built around a common core [4][5]. *Composition* techniques are used to derive products. In approaches based on *negative variability*, a *superimposition* of all variants is created — a *multi-variant domain model*. The derivation of products is achieved by removing all fragments of artefacts implementing features which are *not* contained in the specific feature configuration [6][7]. The toolchain "Features and Mappings in Lucid Evolution" (FAMILE) [8][9], which is used in this paper, belongs to the latter category.

Model-driven Software Engineering (MDSE) [10] puts strong emphasis on the development of high-level models rather than on the source code. Models are not considered as documentation or as informal guidelines how to program the actual system. In contrast, models have a well-defined syntax and semantics. Moreover, MDSE aims at the development of *executable* models. The Eclipse Modeling Framework (EMF) [11] has been established as an extensible platform for the development of MDSE applications. It is based on the Ecore metamodel which is compatible with the OMG Meta Object Facility (MOF) specification [12]. Ideally, software engineers operate only on the level of models such that there is no need to inspect or edit the actual source code, which is generated from the models automatically. However, practical experiences have shown that language-specific adaptations to the generated source code are frequently necessary. In EMF, for instance, only structure is modeled by means of class diagrams, whereas behavior is described by modifications to the generated source code.

In the past, several approaches have been taken in combining SPLE and MDSE to get the best out of both worlds. Both software engineering techniques consider models as primary artefacts: Feature models [3] are used in SPLE to capture the commonalities and differences of a product line, whereas Unified Modeling Language (UML) models [13] or domain-specific models are used in MDSE to describe the software system at a higher level of abstraction. The resulting integrating discipline, Model-Driven Software Product Line Engineering (MDPLE), operates at a higher level of abstraction. The upcoming MDPLE approach has been successfully applied in several case studies, including MOD2-SCM [14], a model-driven product line for software configuration systems.

In this paper, requirements, concepts and limitations of tool support for *heterogeneous* product lines are discussed. Here, the term 'heterogeneity' means that (a) artefacts are distributed over multiple resources, (b) the underlying data format of artefacts may differ (e.g., text files or XMI files), (c) in the case of models, the metamodel may vary, and (d) variability among different resources may be expressed by a shared variability model that uses a common variability mechanism. Based upon these assumptions, several conceptual extensions to MDPLE frameworks are developed, which are implemented in the form of extensions to the toolchain FAMILE as a proof of concept.

The paper is structured as follows: After clarifying the contribution (Section II), the state of the art of homogeneous SPLE tools is outlined in Section IV. Section III discusses related work, before a brief introduction of the running example is given in Section V. Section VI explains the new concepts

 ISBN: 978-1-61208-367-4

introduced for the support of heterogeneous product lines. In Section VII, the example is revisited in order to demonstrate the heterogeneous extension to the MDPLE toolchain FAMILE on a product line for graphs, which has been modeled using Eclipse Modeling Technology (EMF and the Graphical Modeling Framework (GMF) [15]). Both the toolchain and the running example project may be retrieved via an Eclipse update site (http://btn1x4.inf.uni-bayreuth.de/famile2/update). Section VIII concludes the paper.

II. Challenges and Contribution

Heterogeneous software projects consist of a variety of interconnected resources of different types. Different representations may be used for requirements engineering, analysis and design. The generated source code is typically expressed in a general purpose language, e.g., Java, and extended with language-specific – mostly behavioral – components. Furthermore, a software project contains a set of configuration files such as build scripts, which are typically represented in plain text or XML format. In order to adequately handle variability of the overall software project, all these different artefacts need to be subject to variability management.

In its current state, *tool support* for model-driven product line engineering does not adequately address heterogeneous software projects (see Section III). In particular, the following new challenges arise for SPLE tools:

(a) They should ensure the consistency of *cross-resource links* between different artefacts.

(b) The *level of abstraction* needs to be variable, i.e., the tool should be able to operate both at the modeling and at the source code level.

(c) Different artefacts are based on different formalisms, e.g., metamodels or language grammars. In the special case of models, supporting a mixture of different metamodels requires adequate tool support.

(d) All artefacts must be handled by a *uniform variability mechanism* (e.g., a common feature model) in order to allow for product configuration in a single step.

In this paper, an approach to heterogeneous SPL development is presented, which advances the state of the art by the following conceptual contributions:

(a) **Multi-resource artefacts** Heterogeneous projects consist of inter-related models created for different development tasks such as requirements engineering or testing. The referential integrity among these inter-related models is maintained during product derivation.

(b) **Heterogeneous artefact types** The approach presented here can handle product lines composed from different kinds of artefacts. Technically, an abstraction from different resource types is conducted by representing them as EMF models.

(c) **Variable metamodels** In the special case of models, the approach presented here does not assume a specific metamodel but allows an arbitrary mixture of models which may be instances of any Ecore-based metamodel(s).

(d) **Common variability mechanism** In the original version of FAMILE, the variability mechanism of feature models has been applied to single-resource EMF models. The presented approach allows for an extension of the product space to almost arbitrary resources. All artefacts are managed by a unique feature model.

These conceptual contributions will be demonstrated by the example of a proof-of-concept implementation that provides an extension to the FAMILE toolchain [8][9]. The extended version of FAMILE can deal with plain text files, XML files, Java source code files, arbitrary EMF models, and further types of resources. This way, variability within complete Eclipse projects may be managed. Internally, all artefacts, even plain text and XML files, are represented as EMF models. In Section VI, tool support is discussed in detail.

III. Related Work

Many different tools and approaches have been published in the last few years, which address (model-driven) software product line development. Due to space restrictions, the focus of this comparison lies on support for heterogeneous software projects, using the definition of heterogeneity given in the introduction. Other comparisons of FAMILE and related approaches can be found in [8] and [9].

The tool *fmp2rsm* [16] combines FeaturePlugin [17] with IBM's Rational Software Modeler (RSM), a UML-based modeling tool. The connection of features and domain model elements is realized by embedding the mapping information into the domain model using stereotypes (each feature is represented by its own stereotype), which requires manual extensions to the domain model. While fmp2rsm is limited to the support of RSM models, the approach presented in this paper provides a greater flexibility since the only restriction is that the domain model needs to be Ecore based. Furthermore, the extensions presented in this paper allow to use several domain metamodels within one software product line project.

FeatureMapper [6] is a tool that allows for the mapping of features to Ecore based domain models. Like FAMILE, it follows a very general approach permitting arbitrary Ecore models as domain models. FeatureMapper only allows to map a single (self-contained) domain model, while the work presented in this paper allows to use FAMILE also for software product lines whose multi-variant domain model is composed of artefacts distributed over different resources. Furthermore, the artefacts may be instances of different metamodels.

VML* [4] is a family of languages for variability management in software product lines. It addresses the ability to explicitly express the relationship between feature models and other artefacts of the product line. It can handle any domain model as long as a corresponding VML language exists for it. VML* supports both positive and negative variability as well as any combination thereof, since every action is a small transformation on the core model. As a consequence, the order in which model transformations are executed during product derivation becomes important. So far, VML* is designed to work with text files, provided that a corresponding VML

 ISBN: 978-1-61208-367-4

language exists for it (i.e., a grammar has to be specified). Theoretically, VML languages could be written that work with XMI serializations of the respective models in the example presented in this paper, whereas FAMILE provides generic support for model-driven software development based on Ecore compliant models. In other words, VML* and FAMILE provide similar support for heterogeneous projects, but they operate on different "technological spaces". As a consequence, the example provided in Section VII cannot be realized with VML* easily. In fact, significant effort would be required to create VML languages for the different models involved in the graph product line example as presented here.

MATA [5] is another language which also allows to develop model-driven product lines with UML. It is based on positive variability, which means that, around a common core specified in UML, variant models described in the MATA language are composed to a product specific UML model. Graph transformations based on AGG [18] are used to compose the common core with the single MATA specifications. While MATA is limited to UML, the approach presented in this paper provides support for any Ecore based model and furthermore allows the combination of different domain metamodels within one product line project.

CIDE [7] is a tool for source-code based approaches. It provides a product specific view on the source code, where all source code fragments which are not part of the chosen configuration are omitted. The approach is similar to *#ifdef*-preprocessors known from the C programming language [19]. The difference is that it abstracts from plain text files and works on the abstract syntax tree of the target language instead. In its current state, CIDE provides support for a wide range of different programming languages. Unfortunately, it cannot be used for model-driven development. In contrast, FAMILE provides full-fledged support for model-driven development based on Ecore models. Furthermore, it may also deal with regular Java source code by using the MoDisco [20] framework.

Bühne et al. [21] and Dhungana et al. [22] present approaches for heterogeneous variability modeling, i.e., managing commonalities and differences across *multi product lines*. Dhungana et al. aim at unifying multi product lines which rely on different tools and formalisms for modeling variability. Web services are used for a prototypical implementation. In contrast to the approach presented here, in both approaches, the term 'heterogeneity' concerns different variability models rather than the product space. While Bühne et al. and Dhungana et al. only address variability modeling, the approach presented in this paper covers a larger part of the software life-cycle. Furthermore, FAMILE does not only allow for variability modeling, but also for mapping the variability information to heterogeneous implementation artefacts.

IV. State of the Art: Homogeneous MDPLE tools

This section provides a brief overview on the state of the art of current tools for model-driven product line engineering. The description is confined to approaches based on negative variability. As one representative, the original version of the FAMILE toolchain [8][9] is presented. Current MDPLE tools assist the user in the following tasks:

1) **Definition of a feature model** At the beginning of the domain engineering phase of the product line life-cycle, the problem domain is analyzed and the commonalities and differences are captured in a *feature model* [3]. For feature models, several extensions such as cardinality-based feature modeling [23] have been proposed.
2) **Creation of the domain model** For the construction of a multi-variant domain model, modelers may use their preferred modeling languages and tools. Most MDPLE approaches only support single-resource models. FAMILE requires that the resulting model is an instance of an Ecore metamodel.
3) **Mapping features to model elements** In order to define which parts of the domain model realize which feature, or a combination thereof, MDPLE tools provide different mechanisms to map features to model elements. For this purpose, FAMILE includes the Feature to Domain Mapping Model (F2DMM) editor which supports the process of assigning *feature expressions* – arbitrary propositional formula on the set of features – to particular model elements.
4) **Ensuring the consistency of the product line** The increasing complexity coming with both the size of the multi-variant domain model and the number of features requires sophisticated mechanisms to detect and repair inconsistencies among the product line. In particular, the consistency between (a) the mapping model and the domain model, (b) the feature model and its corresponding feature configurations, and (c) feature expressions and the feature model, must be ensured. Different approaches are described in [23], [24]. FAMILE introduces the concepts of *surrogates* and *propagation strategies* [9] for this purpose.
5) **Definition of feature configurations** As soon as the mapping is complete, MDPLE tools support the creation of *feature configurations*, each describing the characteristics of a member of the software product line. For each feature defined in the feature model, a *selection state* must be provided that determines whether a feature is present in the corresponding product.
6) **Product derivation** A specific product can be derived by applying its corresponding feature configuration to the product line. During the derivation process, the multi-variant domain model is filtered by elements whose assigned feature expressions evaluate to false, i.e., the corresponding features are deselected in the respective feature configuration. In homogeneous MDPLE tools, the result of this operation is a product-specific single-resource model represented in the (previously fixed) domain metamodel.

V. Example: Homogeneous FAMILE Product Line for Graph Metamodels

The following statements refer to the original version of the tool FAMILE as one representative of homogeneous MDPLE tools. Section VI demonstrates how heterogeneous project support is added to the toolchain.

FAMILE itself has been developed using EMF as its technological foundation. A model-driven software product

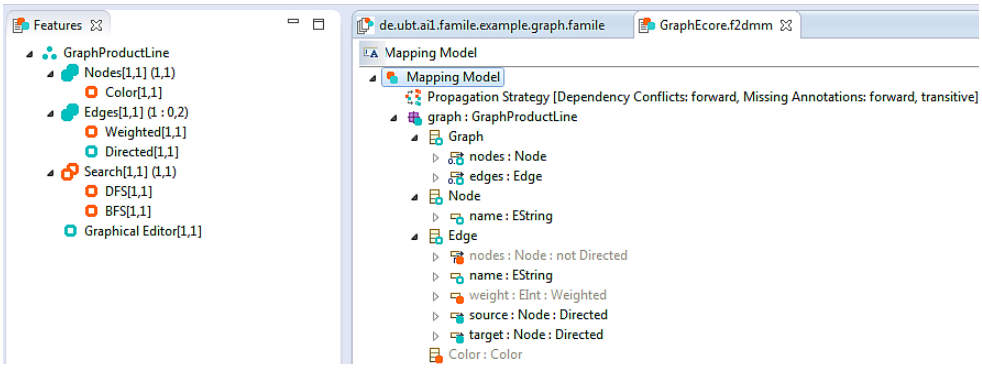


Figure 1. Screenshot of the F2DMM mapping model editor showing the multi-variant domain model of the (homogeneous) graph product line.

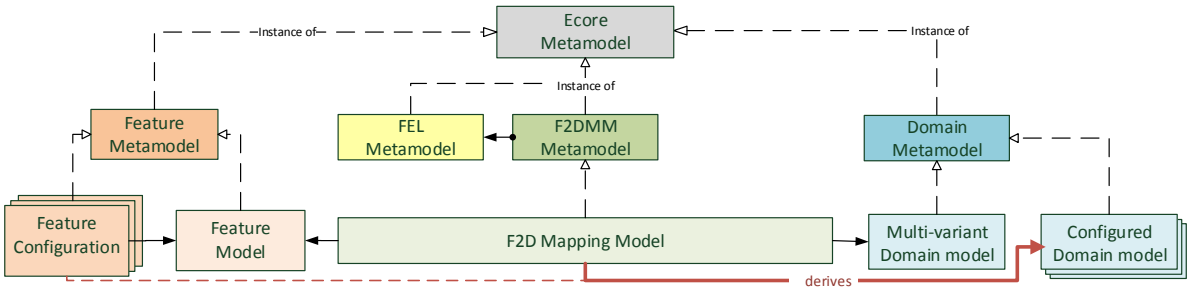


Figure 2. Metamodels and models involved in the original version of FAMILE. Different models are used to map a single-resource multi-variant domain model. All metamodels are based on Ecore.

line developed with FAMILE is spread over multiple EMF resources which are instances of multiple metamodels (cf. Figure 2): Feature models and configurations share a common metamodel which also supports cardinality-based feature modeling. The F2DMM mapping model describes how domain model elements are mapped to features. The domain model is instance of an arbitrary domain metamodel, which is fixed for the mapped resource. It is assumed to be a single-resource entity. The Feature Expression Language (FEL) metamodel describes a textual language for feature expressions [8].

With the F2DMM editor (see Figure 1), the user is assisted in assigning feature expressions to domain model elements. The underlying F2DMM mapping model is constructed automatically and reflects the spanning containment tree structure of the domain model. Using the reflective EMF editing mechanism [11], the F2DMM user interface emulates the reflective EMF tree editor. Optionally, the user may load an example feature configuration already during the mapping process in order to comprehend how feature expressions are evaluated. The screenshot in Figure 1 depicts an example feature configuration in the left pane. Selected features or groups are displayed in cyan, deselected features or groups in orange. The right pane contains the mapping of specific features to artefacts of the multi-variant domain model. Elements are annotated with feature expressions after a colon. The calculated selection states (selected, deselected) are represented in cyan and orange, respectively.

As a demonstrating example within this paper, the *graph product line* example has been adopted, which is frequently used in research papers because it is easy to understand and its size is rather small. In Figure 1, an example feature configuration is loaded that represents a directed graph (with uncolored nodes and unweighted edges) that realizes neither depth-first search nor breadth-first search. The feature "Graphical Editor" will be explained in Section VII, where the example is revisited in the context of heterogeneous product line support.

VI. Support for Heterogeneous Approaches

This section explains how support for heterogeneous model-driven software product lines has been added to the MDPLE tool FAMILE. From a technical point of view, this requires multiple metamodels for the platform and multiple models that describe different artefacts of the product in different stages of the development process (e.g., requirements, static model, implementation). As stated in the introduction, it is assumed that all project artefacts may be expressed using EMF. EMF and its metamodel Ecore are wide-spread in the Eclipse community, thus a large number of potential domain models is addressed. A (non-exhaustive) list may comprise of course Ecore class diagrams, Eclipse UML models [25], Xtext [26] / EMFText [27] grammars and documents, GMF models [15], Acceleo source code generation templates [28], MWE2 Workflow files [29], Xtend specifications [30], domain-specific languages based on Ecore, and many more. Additionally, FAMILE has been applied successfully to Java source code as well. To this end, the MoDisco [20] framework is used, which allows to parse Java source code into a corresponding Java model instance (which is also based on Ecore). MoDisco may be also used to create EMF model instances out of XML files. For plain text files that are not yet mapped by language-specific mechanisms, the new extension framework provides an additional "fall-back" metamodel, which operates on the granularity of text lines. As a consequence, FAMILE may handle arbitrary resource types that may occur within typical

 ISBN: 978-1-61208-367-4

MDSE-related Eclipse projects.

Figure 4 shows the conceptual overview of the new, heterogeneous version of the FAMILE toolchain. A FAMILE model wraps different F2DMM model instances which are used for mapping features to the different (heterogeneous) multi-variant domain model instances. A FAMILE model references a given feature model and one out of an arbitrary number of corresponding feature configurations. Features are mapped to the respective domain artefacts by using a separate mapping model per resource.

A. The FAMILE Metamodel

The specific requirements of heterogeneous modeling projects have been addressed by the FAMILE metamodel and its corresponding instances, which constitute an extension to the F2DMM metamodel, where models have been considered as self-contained single-resource entities [8]. In order to support multiple (EMF-based) resources of different type, the new FAMILE metamodel shown in Figure 3 wraps several instances of the F2DMM metamodel, which still constitutes the core of the extended toolchain.

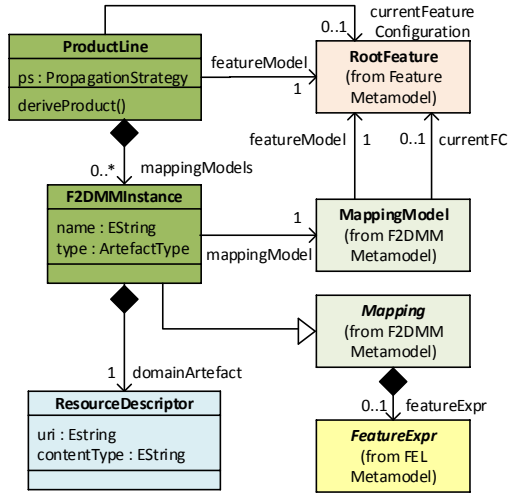


Figure 3. The FAMILE metamodel which is designed to support heterogeneous software product lines.

The FAMILE metamodel defines a logical grouping of inter-related mapping models. The root element – an instance of ProductLine – defines a number of *global project parameters*, being the references to the used feature model and optionally a feature configuration, as well as a *propagation strategy* (used for automatic detection and resolution of inconsistencies; see [9]). FAMILE takes care that global project parameters are kept consistent within different F2DMM resources of the same heterogeneous product line.

A single F2DMM mapping model, which refers to exactly one mapped resource, is represented by F2DMMInstance. This meta-class defines a number of *resource-specific* parameters, such as the name and the *artefact type* (requirements, implementation, test, etc.). Please note that F2DMMInstance extends the abstract meta-class Mapping defined in the F2DMM metamodel, which manages variability by the use of feature expressions and the calculation of selection states [8]. The referenced MappingModel describes the mapping of the specific contents of a mapped resource, e.g., mapped EMF objects in the case of EMF model resources. Furthermore, a contained ResourceDescriptor element describes additional resource-specific parameters, being the relative URI of the mapped resource, as well as its *content type* (plain text, XML, EMF, etc.). The resource containing a multi-variant domain model is referenced by its URI.

Besides the possibility of annotating specific resources of the multi-variant domain model with feature expressions, the presented extension addresses the fact that in heterogeneous projects, *cross-resource links* occur frequently. For instance, in the example in Section VII, elements of an Ecore model are referenced by a corresponding GMF mapping model located in a different resource. During product derivation, these links are detected and resolved automatically in order to meet the requirement of referential integrity across multiple resources. As a result, a derived product will never contain any reference to the multi-variant model.

B. User Interface

The user interface has been extended to support heterogeneous software product lines. A new FAMILE editor manages the mapping for a set of resources rather than single-resource models, which are still covered by the existing F2DMM editor. In addition to the tasks listed in Section IV, the extended FAMILE framework supports the following user interactions (see also example in Section VII):

1) **Adding heterogeneous product line support** An arbitrary Eclipse project containing any kind of resource (e.g., EMF models, source code and documentation) can be provided with the *FAMILE nature*, which adds heterogeneous product line support by automatically creating a FAMILE product line model.
2) **Definition of a global feature model** As soon as the FAMILE nature has been added, the feature model editor is opened automatically and can be used to provide the results of domain analysis. Once a new feature model has been created or an existing feature model has been selected, its contained features may be used in feature expressions annotating corresponding implementation fragments from the multi-variant domain model(s).
3) **Adding variability to resources** Initially, it is assumed that none of the project resources is subject to variability. In order to add variability to a specific resource, the *Add F2DMM Instance* command can be invoked. It will create a new mapping model for the selected resource and append it to the reference mappingModels of the ProductLine instance. Furthermore, global project parameters are transferred to the new F2DMM instance.
4) **Assigning feature expressions to resources** In many cases, variability is achieved at a rather coarse-grained level, having resources rather than objects implement features. The FAMILE editor supports this requirement by the possibility of assigning feature expressions to entire resources.
5) **Applying a feature configuration globally** The command *Set Feature Configuration* allows to change

 ISBN: 978-1-61208-367-4

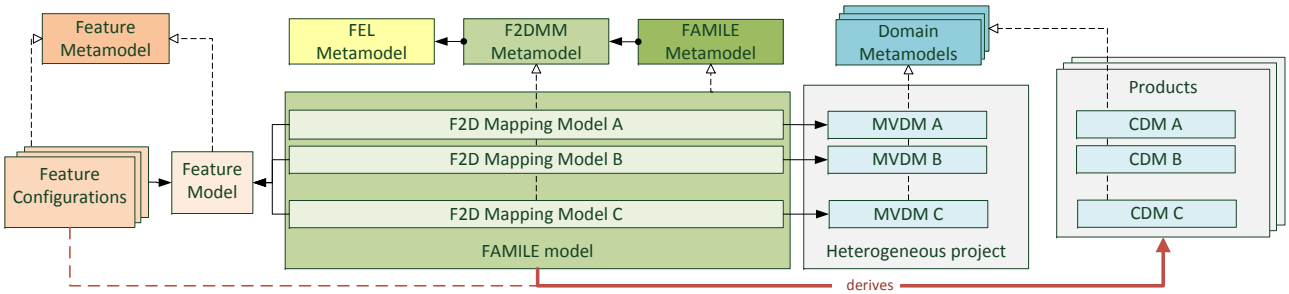


Figure 4. Metamodels and models involved in the extension of FAMILE. Abbreviations: MVDM = multi-variant domain model; CDM = configured domain model.

the current configuration, which will restrict the visible elements/resources in both the F2DMM and the FAMILE editor to elements with a feature expression that satisfies the new configuration. This global project parameter is propagated to all existing F2DMM instances.

6) **Deriving a multi-resource product** After applying a specific feature configuration, a product can be exported. Invoking the *Derive Product* command will prompt the user for a name of the derived Eclipse project. As described above, F2DMM product derivation will be applied to each mapping model covering a resource, keeping cross-resource links consistent. Resources which are not wrapped by any F2DMM instance or which are not annotated with FEL expressions will be copied without any further restriction.

VII. Example Revisited: Heterogeneous Product Line for Graph Metamodels and Editors

To demonstrate FAMILE's support for heterogeneous software projects, the graph product line introduced in Section V has been extended by a graphical editor. To achieve this in a model-driven way, GMF [15] has been used. A screencast demonstrating how to use the extensions for heterogeneous projects provided by FAMILE can be found on the corresponding webpages (http://btn1x4.inf.uni-bayreuth.de/famile/screencasts).

A. GMF Artefacts as a Heterogeneous Set of Multi-Variant Domain Models

Figure 5 depicts the different models involved in the GMF development process. The abstract syntax is defined by an Ecore model, while the editor providing the concrete (graphical) syntax is defined by a graphical definition model, a tooling definition model and a GMF mapping model. The EMF generator model is used to generate Java source code for the abstract syntax while the GMF generator model is responsible for generating the diagram editor's source code. Please note that the screencast does not cover the definition of the models mentioned below. It is assumed that the models describing the abstract and concrete syntax definitions have been created beforehand:

1) **Ecore** The abstract syntax of the graph metamodel has been created in Ecore. As shown in Section V, the F2DMM instance which maps features to the semantic model (abstract syntax).

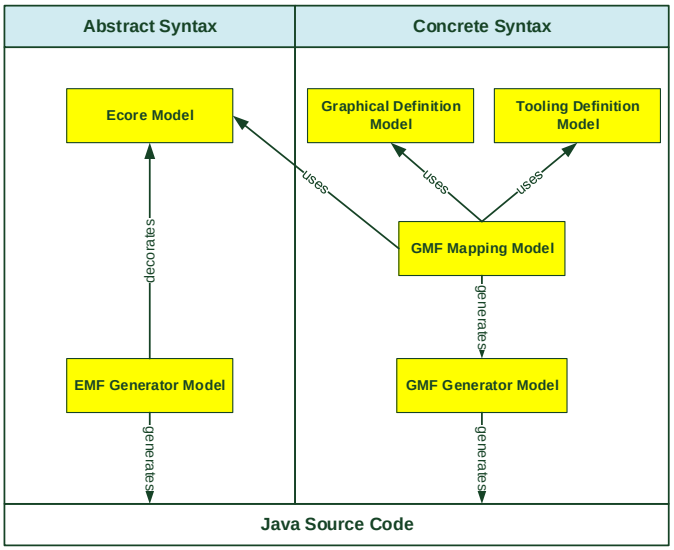


Figure 5. Models involved in the GMF development process.

2) **GMFGraph (Graphical Definition Model)** GMF uses a GMFGraph model to define the graphical representation of the concrete syntax. In case of the example, the visual appearance of nodes and edges of the graph is defined.

3) **GMFTool (Tooling Definition Model)** Every GEF based editor uses a so called palette to drag new elements to the canvas. As GMF is a model-driven extension to GEF, it follows this paradigm. The GMFTooling definition model is used to specify the contents of the editor's tool palette.

4) **GMFMap (GMF Mapping Model)** The models described above (Ecore, GMFGraph and GMFTool) are combined in the GMF mapping model. In this model, a relation between abstract syntax (Ecore) and graphical notation (GMFGraph) is established. Furthermore, the tools (GMFTool) for creating corresponding model elements are linked to those relations. Please note that the GMF mapping model is the central part of the Graphical Modeling Framework. It has nothing in common with the F2DMM mapping model, which is the core of the FAMILE toolchain.

 ISBN: 978-1-61208-367-4

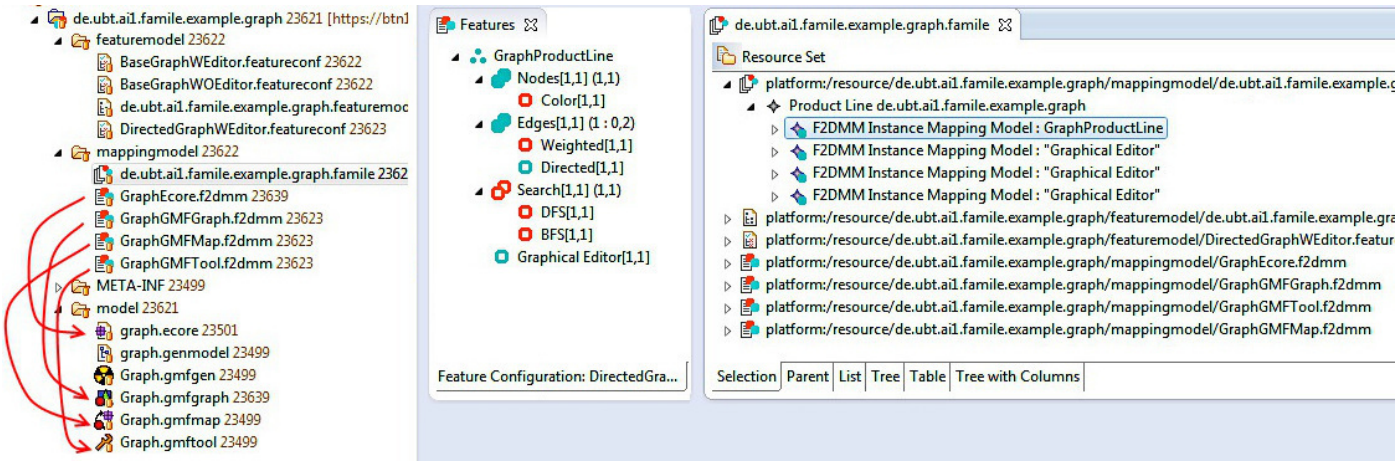


Figure 6. Screenshot of the FAMILE model editor. The left pane shows the feature model and feature configuration. In the main pane, the contents of the FAMILE model are shown.

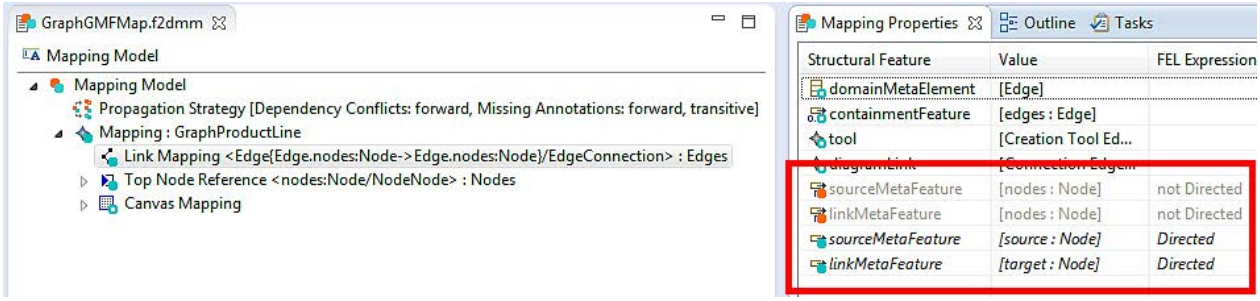


Figure 7. Usage of alternative mappings. The red box depicts where elements of the multi-variant domain model have been virtually extended by alternative mapping values (in italics).

B. *Mapping Heterogeneous Artefacts*

In order to use FAMILE for a (heterogeneous) project, the FAMILE project nature has to be assigned. As a result, an empty feature model and a FAMILE model are created within the project. In the example, the feature model shown in Figure 1 is applied to the entire product line as a global project parameter. In order to map features to corresponding implementation fragments, F2DMM mapping models have to be created for each domain model. In the example, four F2DMM instances have been defined, one for each EMF/GMF resource mentioned above.

Figure 6 depicts the state of the example project after corresponding F2DMM instances have been created. The red arrows in the left part of the figure indicate which domain model resource the corresponding F2DMM models refer to. As one can see, FAMILE model elements may also be annotated with feature expressions. For the example, a feature called *Graphical Editor* has been introduced in order to make the visualization (tree editor vs. graphical editor) of the graph variable. In case this feature is deselected in a respective configuration, it is obvious that the resulting product must not contain the GMF models. As a consequence, the respective F2DMM instances are annotated with the feature expression Graphical Editor, as shown in Figure 6.

Figure 1 has already shown the content of the F2DMM mapping model for the Ecore model which is used to define the abstract syntax of the graph model. Analogously, F2DMM instances for the other required models (GMFGraph, GMFTool and GMFMap) are created. Each model file contains a superimposition of all possible variants. Common approaches using negative variability suffer from restrictions imposed by the respective domain metamodels which usually do not provide adequate support for variability. FAMILE mitigates this restriction by offering the advanced concept of *alternative mappings*. In the example, alternative mappings are used in the Link mapping in the GMFMap model (c.f., Figure 7). In case of an undirected graph, the corresponding graphical editor should just connect two nodes by a solid line. To this end, the underlying semantic model (i.e., the Ecore class model) provides a reference nodes in the class Edge. In contrast, if the feature *Directed* edges is selected, the graphical editor should indicate the direction of the edge connecting two nodes by using an arrow as a target decorator. Furthermore, the semantic model does no longer contain a reference nodes, but instead two single-valued references source and target, which are used to store the corresponding nodes connected by the edge. In GMF, a link mapping requires to specify the corresponding EReferences which are used as the link's source and target. While in the first case, both source and target features in the GMFMap file are set to the EReference nodes, the latter case requires those features to point at the corresponding source and target EReferences.

In this example, FAMILE's alternative mapping capabilities are necessary because the GMF mapping model uses a single-valued EReference to store the sourceMetaFeature

and linkMetaFeature features. In case of undirected edges, the nodes Reference defined in the Ecore model of our graph product line is used. However, in case of directed edges, a distinction between source and target nodes is required. To this end, the Ecore model provides corresponding source and target EReferences in the class Edge (c.f. Figure 1), which have to be used in the GMFMap model instead in case the feature *Directed* is chosen. Figure 7 depicts how this has been solved using FAMILE's alternative mappings [8], which can virtually extend the multi-variant model and thus mitigate the limited variability of the respective domain metamodels.

C. Product Derivation

Once the mapping is completed, specific feature configurations may be used to derive concrete products. In the example, a derived Eclipse project is created which contains the required model files. Please note that the derived project does not include the Ecore and GMF generator models which are required to generate code. Since code generation is always invoked on a *configured* product, this task clearly belongs to the *application engineering* rather than to the *domain engineering* phase.

With the feature configurations provided in the example project, a fully automatic generation of four Eclipse plugin projects may be performed, which differ from each other as follows:

- an EMF tree editor for undirected, unweighted graphs,
- a GMF-based graphical editor for undirected, unweighted graphs,
- a graphical editor for directed, unweighted graphs, and
- a graphical editor for directed, weighted graphs.

Of course this set of feature configurations does not contain all possible combinations of features and it may be extended arbitrarily based on the features defined in the feature model.

D. Outlook: Increasing the Heterogeneity of the Example Project

The example described in this section has been conducted using only EMF-compatible resources as artefacts. All models involved in the GMF development process, i.e., the Ecore domain model, the Graphical Definition Model, the Tooling Definition Model, as well as the GMF Mapping Model, are instances of different Ecore-based metamodels. In the current state of the project, these models constitute the adequate level of abstraction for variability management. However, it might become necessary to define additional F2DMM mapping models for non-EMF resources in addition, for different reasons:

- With Ecore, only *structure* may be modeled in the form of class diagrams. For the *behavioral* part, modifications to the generated Java source code might become necessary. In order to map specific parts of the source code, e.g., specific method bodies, to features, additional F2DMM mapping models may be added for the respective Java files. For this purpose, Java constructs are internally mapped to EMF models using the MoDisco framework, as described in Section VI. Please note that using the current FAMILE extension, the user may annotate Java elements directly in the standard Java text editor.
- The file `plugin.properties` in the Eclipse project contains language-specific UI string constants, each declared in a respective text line. Currently, the generated Editor displays UI elements in English. However, if support for different languages is desired, one may add an additional F2DMM mapping model for the properties file, and corresponding features for each additional language to the feature model. The mapping may be adequately managed by means of a per-line mapping, using the "fall-back" EMF representation for plain text files (see Section VI).
- The file `plugin.xml` defines plugin extensions which are used to integrate the generated editor with the Eclipse platform. By adding an F2DMM mapping model and corresponding features, variability may be added to the plugin's runtime configuration, i.e., in order to make the editor's icon, label, or file extension depend on specific feature configurations. Assuming that no EMF-compatible metamodel for Eclipse plugin files is defined, the "fall-back" EMF representation for XML files (see Section VI) may be used.

VIII. Conclusion and Future Work

In this paper, requirements, concepts and limitations with respect to tool support for heterogeneous model-driven software product lines have been discussed. The approach presented in this paper solves a significant gap in the tool support for model-driven development of software product lines, whose artefacts are heterogeneous in terms of the used metamodels as well as in containing artefacts like text files or XML documents. As a proof of concept, an implementation of an extension to the FAMILE toolchain was shown.

Usually, (model-driven) software projects do not only consist of one single model. In contrast, different models and metamodels are involved. The main challenges of heterogeneous SPLE tool support are (a) to cope with different levels of abstractions (models and source code / plain text files) as well as (b) different forms of representation, (c) to ensure that links between different resources are kept consistent, and (d) to provide a uniform variability mechanism with respect to all project resources.

The approach presented here comes with the assumption that each resource type may be expressed by an EMF model; the new version of FAMILE provides adequate mapping constructs in order to support entire Eclipse projects. Furthermore, the solution to heterogeneous SPLE tooling is to divide a heterogeneous software project into a set of single-resource mapping models, for which adequate SPLE support is already implemented. Links between different models are kept consistent during product derivation. Extensions to the user interface ease the integration of new artefacts into heterogeneous product lines as well as modifications to existing mappings. Furthermore, fallback mechanisms for plain text files and XML files are provided, which also allow to map features to those kinds of artefacts at a lower level of abstraction. A demonstration of the presented approach was given by

 ISBN: 978-1-61208-367-4

applying the heterogeneous FAMILE toolchain to a product line for graph metamodels and editors, which manages an entire Eclipse plug-in project.

Current and future work addresses a case study which is carried out in the field of robotics [31][32]. Although first results produced by the old (homogeneous) version of the FAMILE toolchain are very promising, it is expected that a significant gain in productivity is achieved by exploiting the new, heterogeneous approach. Future work on the tool comprises a better integration of the mapping assistant into the user interface of Xtext-generated textual editors. Furthermore, work in progress addresses extensions to the MoDisco framework in order to provide support for other programming languages like C++ or C#.

REFERENCES

[1] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, Boston, MA, 2001.

[2] K. Pohl, G. Böckle, and F. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Berlin, Germany: Springer Verlag, 2005.

[3] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Carnegie-Mellon University, Software Engineering Institute, Tech. Rep. CMU/SEI-90-TR-21, Nov. 1990.

[4] S. Zschaler, P. Sánchez, J. Santos, M. Alférez, A. Rashid, L. Fuentes, A. Moreira, J. Araújo, and U. Kulesza, "VML* - A Family of Languages for Variability Management in Software Product Lines," in *Software Language Engineering*, ser. Lecture Notes in Computer Science, M. van den Brand, D. Gaevic, and J. Gray, Eds. Denver, CO, USA: Springer Berlin / Heidelberg, 2010, vol. 5969, pp. 82–102.

[5] J. Whittle, P. Jayaraman, A. Elkhodary, A. Moreira, and J. Arajo, "MATA: A Unified Approach for Composing UML Aspect Models Based on Graph Transformation," in *Transactions on Aspect-Oriented Software Development VI*, ser. Lecture Notes in Computer Science, S. Katz, H. Ossher, R. France, and J.-M. Jzquel, Eds. Springer Berlin / Heidelberg, 2009, vol. 5560, pp. 191–237.

[6] F. Heidenreich, J. Kopcsek, and C. Wende, "FeatureMapper: Mapping features to models," in *Companion Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*, Leipzig, Germany, May 2008, pp. 943–944.

[7] C. Kästner, S. Apel, S. Trujillo, M. Kuhlemann, and D. S. Batory, "Guaranteeing syntactic correctness for all product line variants: A language-independent approach," in *TOOLS (47)*, ser. Lecture Notes in Business Information Processing, M. Oriol and B. Meyer, Eds., vol. 33. Springer, 2009, pp. 175–194.

[8] T. Buchmann and F. Schwägerl, "FAMILE: tool support for evolving model-driven product lines," in *Joint Proceedings of co-located Events at the 8th European Conference on Modelling Foundations and Applications*, ser. CEUR WS, H. Störrle, G. Botterweck, M. Bourdells, D. Kolovos, R. Paige, E. Roubtsova, J. Rubin, and J.-P. Tolvanen, Eds. Building 321, DK-2800 Kongens Lyngby: Technical University of Denmark (DTU), Jul. 2012, pp. 59–62.

[9] T. Buchmann and F. Schwägerl, "Ensuring well-formedness of configured domain models in model-driven product lines based on negative variability," in *Proceedings of the 4th International Workshop on Feature-Oriented Software Development*, ser. FOSD 2012. New York, NY, USA: ACM, 2012, pp. 37–44.

[10] M. Völter, T. Stahl, J. Bettin, A. Haase, and S. Helsen, *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006.

[11] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF Eclipse Modeling Framework*, 2nd ed., ser. The Eclipse Series. Boston, MA: Addison-Wesley, 2009.

[12] OMG, *Meta Object Facility (MOF) Core*, formal/2011-08-07 ed., Object Management Group, Needham, MA, Aug. 2011.

[13] ——, *UML Superstructure*, formal/2011-08-06 ed., Object Management Group, Needham, MA, Aug. 2011.

[14] T. Buchmann, A. Dotor, and B. Westfechtel, "Mod2-scm: A model-driven product line for software configuration management systems," *Information and Software Technology*, 2012, http://dx.doi.org/10.1016/j.infsof.2012.07.010. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2012.07.010

[15] R. C. Gronback, *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*, 1st ed., ser. The Eclipse Series. Boston, MA: Addison-Wesley, 2009.

[16] "fmp2rsm project," http://gsd.uwaterloo.ca/fmp2rsm, accessed: 2014-07-15.

[17] M. Antkiewicz and K. Czarnecki, "FeaturePlugin: Feature modeling plug-in for Eclipse," in *Proceedings of the 2004 OOPSLA Workshop on Eclipse Technology eXchange (eclipse'04)*, New York, NY, 2004, pp. 67–72.

[18] G. Taentzer, "AGG: A Graph Transformation Environment for Modeling and Validation of Software," in *Applications of Graph Transformations with Industrial Relevance*, ser. Lecture Notes in Computer Science, J. Pfaltz, M. Nagl, and B. Böhlen, Eds. Charlottesville, VA, USA: Springer Berlin / Heidelberg, 2004, vol. 3062, pp. 446–453.

[19] B. W. Kernighan, *The C Programming Language*, 2nd ed., D. M. Ritchie, Ed. Prentice Hall Professional Technical Reference, 1988.

[20] H. Bruneliere, J. Cabot, F. Jouault, and F. Madiot, "MoDisco: a generic and extensible framework for model driven reverse engineering," in *Proceedings of the IEEE/ACM International Conference on Automated software engineering (ASE 2010)*, Antwerp, Belgium, 2010, pp. 173–174.

[21] S. Bühne, K. Lauenroth, and K. Pohl, "Modelling requirements variability across product lines," in *RE*. IEEE Computer Society, 2005, pp. 41–52.

[22] D. Dhungana, D. Seichter, G. Botterweck, R. Rabiser, P. Grünbacher, D. Benavides, and J. A. Galindo, "Configuration of multi product lines by bridging heterogeneous variability modeling approaches," in *SPLC*, E. S. de Almeida, T. Kishi, C. Schwanninger, I. John, and K. Schmid, Eds. IEEE, 2011, pp. 120–129.

[23] K. Czarnecki, S. Helsen, and U. W. Eisenecker, "Formalizing cardinality-based feature models and their specialization," *Software Process: Improvement and Practice*, vol. 10, no. 1, pp. 7–29, 2005.

[24] F. Heidenreich, "Towards systematic ensuring well-formedness of software product lines," in *Proceedings of the 1st Workshop on Feature-Oriented Software Development*. Denver, CO, USA: ACM, Oct. 2009, pp. 69–74.

[25] "Eclipse UML2 Project," http://www.eclipse.org/modeling/mdt/?project=uml2, accessed: 2014-07-15.

[26] "Xtext project," http://www.eclipse.org/Xtext, accessed: 2014-07-15.

[27] "EMFText Project," http://www.emftext.org, accessed: 2014-07-15.

[28] "Acceleo project," http://www.eclipse.org/acceleo, accessed: 2014-07-15.

[29] "MWE2 Project," http://www.eclipse.org/modeling/emft/?project=mwe, accessed: 2014-07-15.

[30] "Xtend project," http://www.eclipse.org/xtend, accessed: 2014-07-15.

[31] J. Baumgartl, T. Buchmann, D. Henrich, and B. Westfechtel, "Towards easy robot programming: Using dsls, code generators and software product lines," in *Proceedings of the 8th International Conference on Software Paradigm Trends (ICSOFT 2013)*, J. Cordeiro, D. Marca, and M. van Sinderen, Eds. ScitePress, Jul. 2013, pp. 548–554.

[32] T. Buchmann, J. Baumgartl, D. Henrich, and B. Westfechtel, "Towards a domain-specific language for pick-and-place applications," in *Proceedings of the Fourth International Workshop on Domain-Specific Languages and Models for Robotic Systems (DSLRob 2013).*, U. P. S. Christian Schlegel and S. Stinckwich, Eds. arXiv.org, 2013. [Online]. Available: http://arxiv.org/abs/1401.1376

 ISBN: 978-1-61208-367-4

System Composition Using Petri Nets and DEVS Formalisms

Radek Kočí and Vladimír Janoušek

Brno University of Technology, Faculty of Information Technology,
IT4Innovations Centre of Excellence
Brno, Czech Republic
{koci,janousek}@fit.vutbr.cz

***Abstract*—This paper is part of the work dealing with system developoment and deployment, where the system behavior should be modeled by formalisms allowing to define workflow scenarios and offering an interface for workflows synchronization. One such formalism is represented by Object Oriented Petri Nets (OOPN). OOPN are based on well-known class-based approach enriched by concurrency. Nevertheless, OOPN lacks one important element—a hierarchy followed by a simple way to model items exchanges on the fly. Therefore, the formalism of Discrete Event System Specification (DEVS) is taken into account. In the presented approach, the OOPN model is split up into DEVS components. Each component can be coupled with another component through the same compatible interface. A combination of OOPN and DEVS formalisms is used to compose the system using DEVS-based components, where each such component is modeled by means of OOPN. It preserves the advantages of using OOPN for behavior modeling and makes it possible to hierarchize models. The paper deals with the combination of both formalisms and compares the classic object approach to the component approach for system composition.**

Keywords-Object Oriented Petri Nets; DEVS; system composition; data passing.

I. Introduction

The paper is part of the work dealing with system developoment and deployment [1][2][3], where the system is modeled, as well as implemented, by means of formal models. Present methods use various models in analysis and design phases, whereas these models usually serve as a system documentation rather than real models of the system under development. The system is then implemented in accordance with these models, whereas the code is either generated from models or implemented manually. Unfortunately, many implementation differ from the designed models because of changes created during the system debugging and improvement. Consequently, models become out of date and useless.

To solve this problem, the methodologies and approaches commonly known as Model-Driven Software Development were investigated and developed for many years [4][5]. These methods use executable models, e.g., Executable Unified Model Language (ExecUML) [6] in Model Driven Architecture methodology [7], which allow to test systems using models. Models are then transformed into code, but the resulted code has to often be finalized manually and the problem of imprecision between models and transformed code remains unchanged.

The system development methodology, which makes a base of the presented work, uses formal models for system description, as well as system implementation. Therefore, there is no need to transform models. Moreover, the system is developed using different kinds of models in simulation, i.e., it is possible to test systems in any state at any time. The combination of formalisms allows to derive benefits from their different features. This paper deals with two formalisms—Object Oriented Petri Nets (OOPN) [8], [9] and Discrete Event System Specification (DEVS) [10]. It preserves the advantages of using OOPN for behavior modeling and makes it possible to compose systems using DEVS-based components. This combination has been already used in previous works [2][11][12] as it is, without an analysis of its features and usefulness. This paper puts an accent on the ability of that concepts to model a system composition and its usability is demonstrated ising an example, which was defined in [11].

The paper is organized as follows. Section II deals with related work. Then, we briefly introduce the system in simulation concept in Section III and the used formalisms of OOPN and DEVS in Section IV. The different principles of system composition will be described in Section V, followed by analysis of system elements communication in Section VI. The usability of the presented approach is demonstrated in Section VII and the summarization and future work will be described in Section VIII.

II. Related Work

There are many works dealing with similar problems in the field of the design of control or embedded systems. The common feature is to use formal system (language, models, etc.) to software design and testing. There are two main motivations of formal system usage. First, to build and maintain control of the system in a quite fast and inuitive way. The High-level languages, especially based on Petri Nets, are used in this way. For example, the RoboGraph framework [13] for the robot application control uses hierarchical binary Petri nets for middleware implementation. In the area of

 ISBN: 978-1-61208-367-4

embedded systems, we can mentioned the work by Rust et al. [14], which uses Timed Petri Nets for the synthesis of control software by generating C-code, the work based on Sequential Function Charts [15], or the work based on the formalism of Nets-Within-Nets (NwN) [16][17][18].

These tools and works allow to *model* systems using a combination of different formalisms, but do not allow to use formal models in system *implementation*. The formalism of NwN is closest to the formalism of OOPN, but OOPN fully support an integration of formal description and programming language, which facilitates, e.g., reality-in-the-loop simulation or usage of formal models in the target application. The proposed approach allows to use of formal models to design, analyze and program applications, including a combination of simulated and real components. The main advantages are the following: there is no need for code generation, and the same formalisms and methods are used for further investigation of deployed systems.

III. System in Simulation Concept

The basic principles of the system development methodology [3][11] will be introduced in this section. The methodology supposes that the system is developed in the simulation; this concept will be outlined, too.

A. System Development Methodology

The modeling process is split up into three basic phases—identification of model elements, modeling the system behavior, and modeling the system architecture. The basic model elements are *subjects*, *roles*, and *activities*. The subject represents a data unit, e.g., the user working with the system or an individual element in the system. Each subject acts through its role. One subject can have more roles, e.g., the user can act as a reviewer, as well as a participant, in the conference review system. The activity represents the system functionality and is modeled by workflow scenarions. To model each such element, the formalism of OOPN is used. It can also be modeled by any other formalism allowing to define workflow scenarios and offering an interface for workflows synchronization, e.g., statecharts, activity diagrams, or other kind of Petri nets. The system architecture is modeled by classes that can be coupled into components using the formalism of DEVS.

B. Application Framework For System in Simulation

The used methodology [11] supposes that the system is being developed in the simulation. The development process starts with the empty simulation (simulation containing no model elements). Subsequently, in every subsequent step, model elements are being created, modified, or exchanged within the simulation. The simulation can be suspended, resumed, or restarted at any time, so that designers are able to test system behavior immediately, after each change. The model can be tested in real conditions, too. Therefore, a possibility to communicate with elements of product environment has to be ensured. The product environment is the target system where the developed model has to work.

The presented concept has to be supported by an application framework allowing to model the system, to simulate designed models, and to manipulate with models during the simulation. The application framework has to fulfil three basic requirements. First, to link models and product environment. Second, to work with models in simulations. Third, to exchange elements of models *on the fly*—the model elements should be exchanged with no changes in the depending model elements.

The application framework PNtalk, which satisfies previously listed requirements, has been developed [19]. Since the framework is implemented in Smalltalk [20], the objects of the OOPN formalism are directly available in the Smalltalk application and Smalltalk objects are directly available in the PNtalk framework. Nevertheless, OOPN objects can be linked to objects of any languages or formalisms allowing message passing.

Second, the PNtalk framework allows to execute models in different simulation modes that are suitable for design, testing, hardware/software-in-the-loop simulation, and system deployment. Using simulation allows, among others, to suspend (i.e., to exclude from the execution), to modify, or to exchange chosen parts of the model. By this point, we came in on the third requirement—a possibility to exchange model parts on the fly (any time during the system simulation). Therefore, the formalism of DEVS is taken into account. DEVS offers component approach allowing for wrapping an other kind of formalisms. The combination of OOPN and DEVS formalisms preserves the advantages of using OOPN for behavior modeling and makes it possible to hierarchize models. It allows the designer to derive benefits from component exchanges instead of object exchanges.

IV. Formal Models

We will briefly introduce the formalisms of OOPN and DEVS that make a base of the system development methodology [3][11].

A. Formalism of Object Oriented Petri Nets

The formalism of OOPN [21] is based on the well-known class-based approach. All objects are instances of classes, every computation is realized by message sending, and variables contain references to objects. This kind of object-orientation is enriched by concurrency. OOPN objects offer reentrant services to other objects and, at the same time, they can perform their own independent activities. The services provided by the objects, as well as the autonomous activities of the objects are described by means of high-level Petri nets—services by *method nets*, object activities by *object nets*.

 ISBN: 978-1-61208-367-4

The formalism of OOPN contains important elements allowing to test object states (*predicates*) and to manipulate object state (*synchronous ports*) with no need to instantiate method nets. Object state testing can be negative (*negative predicates*) or positive (*synchronous ports*). *Synchronous ports* are special (virtual) transitions that cannot fire alone but need to be dynamically fused to some other transitions the synchronous port is called from (via message sending). *Negative predicates* are special variants of synchronous ports having inverted semantics—the calling transition is fireable if the negative predicate is not fireable.

For the sake of notation simplicity, we introduce the formal notation for following relationships. The term @ represents the relationship *is an instance of*. For example, $(a, C1) \in @$ means that a is the instance of the class named $C1$. We will write this relation in the form $a@C1$. If the instance identifier is not important, we will type only $@C1$. The terms $\overset{\mathcal{M}}{\rightharpoonup}$ and $\overset{\mathcal{M}}{\leftharpoonup}$ represent the relationship *contains a reference to*. For example, $a_1@A \overset{\mathcal{M}}{\rightharpoonup} a_2@B$ means that $a_1@A$ contains a reference to $a_2@B$ and $a_1@A \overset{\mathcal{M}}{\leftharpoonup} a_2@B$ means that $a_2@B$ contains a reference to $a_1@A$. If there are both relationships on the same elements, we will write $a_1@A \overset{\mathcal{M}}{\rightleftharpoons} a_2@B$.

B. Formalism of DEVS

The formalism of DEVS [10] can represent any system whose input/output behavior can be described as a sequence of events. The atomic DEVS model is specified as a structure M containing sets of states S, input and output event values X and Y, internal transition function δ_{int}, external transition function δ_{ext}, output function λ, and time advance function ta. These functions describe the behavior of the component.

This way, we can describe atomic models. Atomic models can be coupled together to form a coupled model CM. The later model can be employed itself as a component of a larger model. This way, the DEVS formalism brings a hierarchical component architecture. Sets S, X, Y are to be considered as structured sets. It allows to use multiple variables for specification of a state; we can use a concept of input and output ports for input and output events specification, as well as for coupling specification. In another words, components are connected by means of ports and event values are carried through these ports. We will denote input port by the notation $component\_name \oplus port\_name$ and output port by the notation $component\_name \ominus port\_name$.

As with the object approach, we will define the formal notation of DEVS components. Since the component represents the model description (cf. classes), as well as its executable form (cf. objects as instances of classes), there is no means for a notion *to be an instance*. The new component having the same structure and behavior of existing one can be created by *clonning* that component. To differ from the notation *contains a reference to* $\overset{\mathcal{M}}{\rightleftharpoons}$, we define the relationship *linked with* meaning that the component is linked with another one through their ports. This relationship will be represented by terms *left link* $\overset{\mathcal{D}}{\rightharpoonup}$, *right link* $\overset{\mathcal{D}}{\leftharpoonup}$, and *link* $\overset{\mathcal{D}}{\rightleftharpoons}$. For example, $c_1 \overset{\mathcal{D}}{\rightharpoonup} c_2$ (or $c_1 \overset{\mathcal{D}}{\leftharpoonup} c_2$) establishes a channel for data transmission from the component c_1 to the component c_2 (or from c_2 to c_1). The link $\overset{\mathcal{D}}{\rightleftharpoons}$ means there are both (left and right) links. To specify ports, we will write $c_1 \ominus port_1 \overset{\mathcal{D}}{\rightharpoonup} c_2 \oplus port_2$.

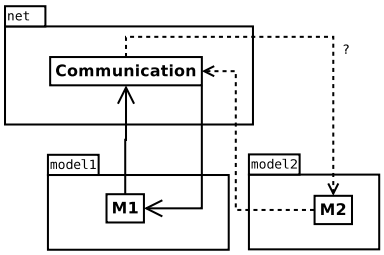


Figure 1. Packages with bidirectional relationship.

C. Combination of OOPN and DEVS Formalisms

The DEVS formalism offers a component approach, allowing to wrap other kinds of formalisms, so that each such formalism is interpreted by its simulator and simulators communicate with each other by means of the compatible interface. The OOPN model is then split up into components linked together by the compatible interface. Let $M_{PN} = (M, \Pi, map_{inp}, map_{out})$ be a DEVS component M, which wraps an OOPN model Π. The model Π defines an initial class c_0, which is instantiated immediately the component M_{PN} is created. Functions map_{inp} and map_{out} map ports and places of the object net of the initial class c_0. The mapped places then serve as input or output ports of the component, i.e., they are part of the component interface.

V. System Composition

The different principles of system composition will be described in this section. First, we describe the classic object oriented approach defining *packages*, where the interface is build up from classes or objects. Second, we describe the DEVS approach defining *components*, where the interface is build up from *ports*.

A. Packages Composition

In the classic approach, the interface of each package is built up from *classes* and their *operations*. Relationships between two packages should be only unidirectional—if there are bidirectional relationships, packages cannot be simply replaced by other packages. The example is shown in Figure 1. There are two packages `net` and `model1`; the class `model1.M1` needs to use the class

`net.Communication` and `net.Communication` notifies `model1.M1` about incomming events—the relationship is bidirectional. If the package `net` would be used with another package (e.g., `model2`), there is a problem how to represent the association from `net.Communication` to `model2.M2`. The only way is *to change* the package `net`.

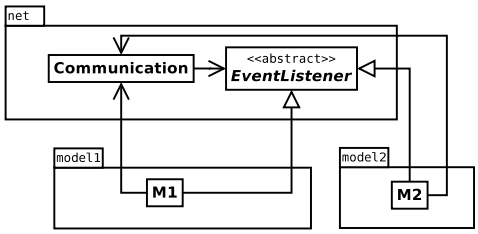


Figure 2. Packages with unidirectional relationship.

Figure 2 shows a solution of the previous situation—the class `net.Communication` depends on the newly created class `net.EventListener` and `model1.M1` is derived from it—the relationship is unidirectional (only from `model1` to `net`). If the package `net` would be used with another package, e.g., `model2`, this package only uses classes from `net`—no changes are needed. It is an application of known *Dependency Inversion Principle*.

B. Components Composition

In DEVS approach, the component interface is built up from *ports*. Relationships between two packages do not need to be only unidirectional; components can be simply replaced by other components in both cases. The example is shown in Figure 3. There are two components `net` and `model1`; the component `model1` sends commands to the component `net` and `net` notifies `model1` about incomming evets—the relationship is bidirectional.

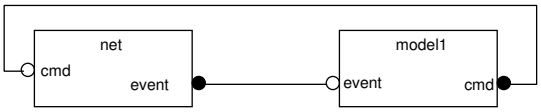


Figure 3. Components with bidirectional relationship.

If the package `net` would be used with another component (e.g., `model2`), there is no problem how to do it—the new component is simply re-connected through ports (see Figure 4).

VI. COMMUNICATON

The difference between objects and components replacement *on the fly* will be taken into account in this section.

A. Message Passing

In the classic approach, the package communication is provided by *message passing*. The object from one package (*client*) sends a message to the object from second package (*server*), whereas the client usually waits for an answer (until the message is processed—synchronous communication).

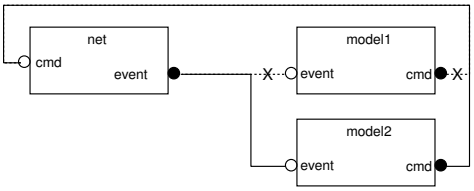


Figure 4. Components composition through ports with component changing.

Figure 5a) shows the communication between packages `net` and `model1` through their interfaces. The instance $@net.Communication$ notifies the instance $@model1.M1$ about arrising events and $@model1.M1$ sends commands to $@net.Communication$. Let us assume that the class $Communication$ represents an interface to the real robot and the class $M1$ implements control algorithms. During the simulation, there can arise a need to test another algorithms in the current situation—the simple way is to change the control algorithm *on the fly* and continue in simulation. So, the component $model1$ is exchanged to $model2$; it follows that $@model1.M1$ is removed and $@model2.M2$ is put in its place.

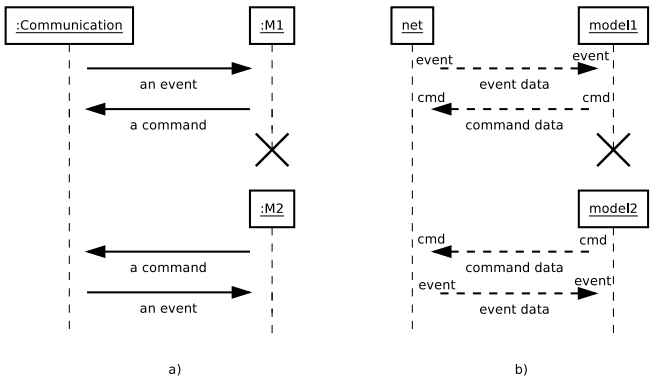


Figure 5. Communication through the object and component interfaces.

The object exchange *on the fly* means that one reference is exchanged to another one. To achieve this goal, the server has to be prepared for such an operation—it has to offer a protocol for attaching and detaching clients. Second, if the new client has a different protocol, it has to be adapted (cf., e.g., the design pattern *Adapter*). Third, if the detached component is in process, e.g., it processes a method which has been called from another object, the problem of its correct removing arrises.

 ISBN: 978-1-61208-367-4

If we get back to the previous example, we make out that the instance of the class `net.Communication` has a reference to the instance of the class `model1.M1` and vice versa—from this point of view, there is a bidirectional relationship $@net.Communication \overset{\mathcal{M}}{\rightleftharpoons} @model1.M1$.

B. Data Passing

In DEVS approach, the component interface is built up from *ports*. Component communication is then provided by *data passing*; the *client* component sends a piece of data to the *server* component, whereas the client usually does not wait for an answer (asynchronous communication).

Figure 5b) shows a data passing between components `net` and `model1` through their interfaces. The component net notifies the component $model1$ about arrising events by carrying a piece of data from $net \ominus event$ to $net \oplus event$. The component $model1$ sends commands to the component net by carrying a piece of data through the data connection $model1 \ominus cmd \xrightarrow{\mathcal{D}} net \oplus cmd$. Since the component represents the model description (cf. classes), as well as its executable form (cf. objects as instances of classes), there is no difference between components replacements during their composition or on the fly.

C. Comparison of Data and Message Passing

Since the Petri nets have good ability of describing processes, they can be used to model the difference between *message passing* and *data passing*. Figure 6a) describes the model of message passing and Figure 6b) describes the model of data passing.

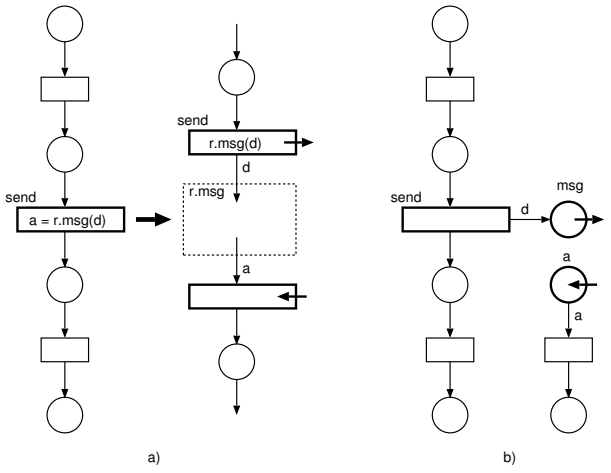


Figure 6. Comparison of message passing and data passing.

Let us investigate Figure 6a). The sequence of unnamed places and transitions represents one thread in the system behavior containing the message passing. It is modeled by the transition named $send$—it sends a message msg to the receiver r with a piece of data d and waits for the answer a. The transition $send$ can be split up into *output transition* (shown with output arrow), which needs to know receiver, message, and data, and *input transition* (shown with input arrow), which waits for the answer, i.e., waits until the called method $r.msg$ is finished.

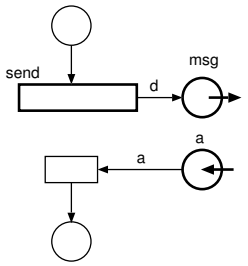


Figure 7. Adjusting data passing to sychronous communication.

Let us investigate Figure 6b). The transition $send$ needs to know only a piece of data d that are put to the *output port* msg (shown with output arrow) representing the message. First, the component does not care about the receiver (anything what is linked) and its interface. Second, the transition $send$ models asynchronous communication (no waiting for the answer). If the component needs to get the answer, it can define an independent thread, which is started by putting the answer to the *input place* a (shown with input arrow). In the case the component needs to wait for the answer, it can be modeled as shown in Figure 7.

VII. Data Passing and System in Simulation Concept

This section demonstrates the usability of the presented approach on a simple example of the robotic system, which has been described in [11]. The robotic system consists of the simulated robot (the data unit modeled by the subject *Device*), its role *Robot*, and one possible scenario of the robot behavior (the activity *Scenario*). These model elements are identified in accordance with development methodology (see Section III).

A. Model of Behavior

We will suppose a very simple activity, which can be described by the following algorithm: (1) the robot is walking; (2) if the robot comes upon to an obstacle, it stops, turns right and tries to walk, (3) if the robot cannot walk, it turns round and tries to walk; (4) if there is no possibility to walk, it stops. The activity net *Scenario* describing the presented behavior is shown in Figure 8.

The activity $@Scenario$ communicates with the object $@Role$, which is initially placed in the place *walking*. The communication is provided using *predicates* that serve for testing (see *isCloseToObstacle* and *isClearRoad*) and *synchronous ports* for action performing (see *stop*, *go*, and *turnRight*).

 ISBN: 978-1-61208-367-4

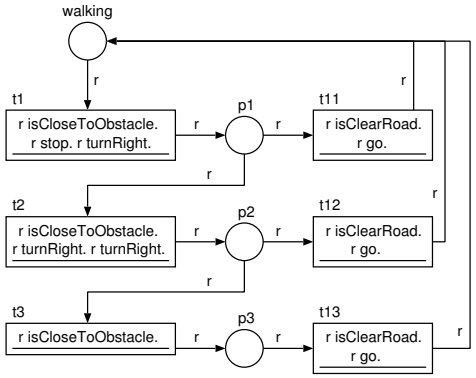


Figure 8. The activity net *Scenario*.

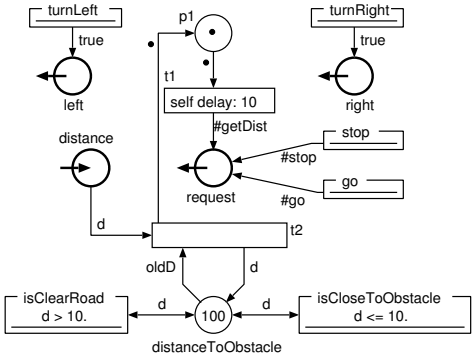


Figure 9. The role *Robot*.

The possible model of the role *Robot* is shown in Figure 9. The role offers information about robot's position by means of predicates *isClearRoad* and *isCloseToObstacle*. Moreover, the role offers synchronous ports *stop*, *go*, *turnRight*, and *turnLeft*, that represent commands forwarded to the subject.

B. Model of Composition

The system is composed of two components that are shown in Figure 10. The component *behavior1* consists of the role *Robot* and the activity *Scenario*—since these objects communicate using *message passing*, they have to be encapsulated into the same component. The component *robot* represents the *subject*, whose realization is schematically depicted in Figure 11.

The communication is provided using *data passing*. The role *Robot* represents the initial class c_0 of the component *behavior1*, so that the object $@Robot$ defines input and output ports. The component interface consists of output ports $\ominus request$, $\ominus left$, and $\ominus right$. We can see that appropriate synchronous ports only put a piece of data to these ports.

The component *robot* has input ports corresponding to the output ports of the component *behavior1*—their interfaces are compatible. Let us investige the following situation.

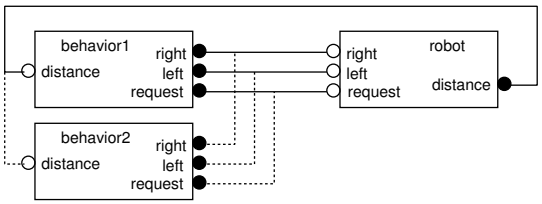


Figure 10. The robot system composition.

The activity $@Scenario$ requests turning right—it calls the synchronous port $@Robot.turnLeft$, which puts a value $true$ to the output port $behavior1 \ominus left$. This value is carried through $behavior1 \ominus left \xrightarrow{\mathcal{D}} robot \oplus left$ to the component *robot*, where performs advisable operations.

The role $@Robot$ checks actual robot's distance to the obstacle every 10 time units by requesting new data—it carries a symbol $\#getDist$ through $behavior1 \ominus request \xrightarrow{\mathcal{D}} robot \oplus request$ to the component *robot*. The component *robot* gets a new information about the distance and carries it back through $robot \ominus distance \xrightarrow{\mathcal{D}} behavior1 \oplus distance$.

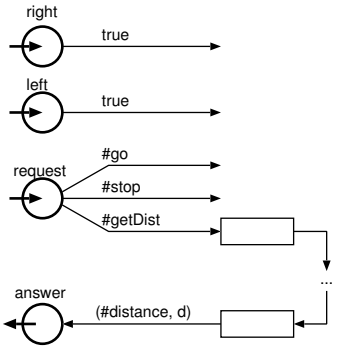


Figure 11. The subject component—an abstract view.

Anytime we need to exchange the model of behavior (for example, the actions change from *turning right* to *turning left*), we simply clone the existing component *behavior1*, the new component named *behavior2* will be created, we modify its realization and connect it through ports (see the component *behavior2* in Figure 10).

VIII. Conclusion and Future Work

This paper dealt with the usability of the OOPN and DEVS formalisms in the system development. The formalism of OOPN allows to define workflow scenarios and offers an interface for workflows synchronization. Nevertheless, it lacks a hierarchy followed by simple way to model items exchanges on the fly. Therefore, it was combined with the formalisms of DEVS, which offers hierarchized component approach—the OOPN model is split up into components linked together by the compatible interface. It preserves

 ISBN: 978-1-61208-367-4

the advantages of using OOPN for behavior modeling and makes it possible to hierarchize models.

This paper is part of the work dealing with system development and deployment using specific methodology and tool support. The application framework PNtalk, which satisfies required features, has been developed [19]. So far, it allows a communication to Smalltalk environment. Nevertheless, it can be linked to objects of any languages or formalisms allowing message passing. We plan to extend PNtalk to the Java and C/C++ platforms.

The proposed approach has one main disadvantage—the usage of application framework interpreting formal models, increases requirements on memory size and system performance. The future research will aim at efficient representation of choosed formal models and interoperability with another product environments. The application framework will be adapted to these conditions having lesser requirement for resources.

ACKNOWLEDGMENT

This work has been supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070) and by BUT FIT grant FIT-S-14-2486.

REFERENCES

[1] R. Kočí and V. Janoušek, "System design with Object oriented Petri nets formalism," in *The Third International Conference on Software Engineering Advances Proceedings ICSEA 2008*. IEEE Computer Society, 2008, pp. 421–426.

[2] R. Kočí and V. Janoušek, "OOPN and DEVS formalisms for system specification and analysis," in *The Fifth International Conference on Software Engineering Advances*. IEEE Computer Society, 2010, pp. 305–310.

[3] R. Kočí and V. Janoušek, "Modeling and simulation-based design using Object-oriented Petri nets: a case study," in *Proceeding of the International Workshop on Petri Nets and Software Engineering 2012*, vol. 851. CEUR, 2012, pp. 253–266.

[4] S. Beydeda, M. Book, and V. Gruhn, *Model-Driven Software Development*. Springer-Verlag, 2005.

[5] M. Broy, J. Gruenbauer, D. Harel, and T. Hoare, Eds., *Engineering Theories of Software Intensive Systems: Proceedings of the NATO Advanced Study Institute*. Kluwer Academic Publishers, 2005.

[6] C. Raistrick, P. Francis, J. Wright, C. Carter, and I. Wilkie, *Model Driven Architecture with Executable UML*. Cambridge University Press, 2004.

[7] D. S. Frankel, *Model Driven Architecture: Applying MDA to Enterprise Computing*, ser. 17 (MS-17). John Wiley & Sons, 2003.

[8] M. Češka, V. Janoušek, and T. Vojnar, *PNtalk — a computerized tool for Object oriented Petri nets modelling*, ser. Lecture Notes in Computer Science. Springer Verlag, 1997, vol. 1333, pp. 591–610.

[9] R. Kočí and V. Janoušek, *Simulation Based Design of Control Systems Using DEVS and Petri Nets*, ser. Lecture Notes in Computer Science. Springer Verlag, 2009, vol. 5717, pp. 849–856.

[10] B. Zeigler, T. Kim, and H. Praehofer, *Theory of Modeling and Simulation*. Academic Press, Inc., London, 2000.

[11] R. Kočí and V. Janoušek, "Object oriented Petri nets in software development and deployment," in *ICSEA 2013, The Eighth International Conference on Software Engineering Advances*. Xpert Publishing Services, 2013, pp. 485–490.

[12] R. Kočí and V. Janoušek, "Towards Design Method Based on Formalisms of Petri Nets, DEVS, and UML," in *ICSEA 2011, The Sixth International Conference on Software Engineering Advances*, 2011, pp. 299–304.

[13] J. L. Fernandez, R. Sanz, E. Paz, and C. Alonso, "Using hierarchical binary Petri nets to build robust mobile robot applications: RoboGraph," in *IEEE International Conference on Robotics and Automation*, 2008, pp. 1372–1377.

[14] C. Rust, F. Stappert, and R. Kunnemeyer, "From Timed Petri nets to interrupt-driven embedded control software," in *International Conference on Computer, Communication and Control Technologies (CCCT 2003)*, 2003, pp. 124–129.

[15] O. Bayo-Puxan, J. Rafecas-Sabate, O. Gomis-Bellmunt, and J. Bergas-Jane, "A GRAFCET-compiler methodology for C-programmed microcontrollers, In Assembly Automation," *Assembly Automation*, vol. 28, no. 1, pp. 55–60, 2008.

[16] R. Valk, "Petri nets as token objects: an introduction to Elementary object nets." in *Jorg Desel, Manuel Silva (eds.): Application and Theory of Petri Nets; Lecture Notes in Computer Science*, vol. 120. Springer-Verlag, 1998.

[17] D. Moldt, "OOA and Petri nets for system specification," in *Object-Oriented Programming and Models of Concurrency*. Italy, 1995.

[18] L. Cabac, M. Duvigneau, D. Moldt, and H. Rölke, "Modeling dynamic architectures using nets-within-nets," in *Applications and Theory of Petri Nets 2005. 26th International Conference, ICATPN 2005, Miami, USA*, 2005, pp. 148–167.

[19] R. Kočí. PNtalk system. [Online]. Available: http://perchta.fit.vutbr.cz/pntalk2k [retrieved: August, 2014]

[20] A. GoldBerk and D. Robson, *Smalltalk 80: The Language*. Addison-Wesley, 1989.

[21] V. Janoušek and R. Kočí, "PNtalk: concurrent language with MOP," in *Proceedings of the CS&P'2003 Workshop*. Warsaw University, Warsawa, PL, 2003, pp. 271–282.

 ISBN: 978-1-61208-367-4

A Prototyping Discipline in OpenUP to Satisfy Wireless Sensor Networks Requirements

Gian Ricardo Berkenbrock
Software/Hardware Integration Lab.
Federal University of Santa Catarina (UFSC)
Joinville, SC, Brazil
Email: gian.rb@ufsc.br

Carla Diacui Medeiros Berkenbrock
Santa Catarina State University (UDESC)
Department of Computer Science
Joinville, SC, Brazil
Email: carla.berkenbrock@udesc.br

Celso Massaki Hirata
Department of Computer Science
Aeronautics Institute of Technology (ITA)
S.J.Campos - SP, Brazil
Email: hirata@ita.br

***Abstract*—Wireless Sensor Networks (WSNs) are used to collect data from different sources and they can be applied in monitoring and instrumentation areas. WSN are highly dependent on application requirements, then one application is hardly equal to another. There is not a specific process to address the development of WSN applications. Open Unified Process is an iterative software development process that is intended to be minimal, complete, and extensible, and because of these features it is a good candidate for WSN application development. However, OpenUP does not support the challenges and requirements of WSN systems, because it does not have specific tasks that consider such requirements. Then, in order to address this lack of support, this paper proposes a prototype discipline that can be integrated into software development process for WSN applications.**

Keywords–Software Engineering; Prototype Discipline; Discrete Simulation.

I. INTRODUCTION

Wireless Sensor Networks (WSNs) are used in environmental monitoring, surveillance of installations or areas, such as, home, border, or building, object tracking, precision agriculture, bridge monitoring, hospital monitoring, and herd monitoring. WSNs are composed of nodes that have capability to sense and communicate over-the-air to each other. The nodes have also processing capability and local storage. In order to deliver the collected data to a base station in a multi-hop scenario, nodes transmit data using ad hoc communication. The nodes have the capability to create the wireless interconnection network, which is needed to delivery the data.

The number of nodes of WSNs can change from a few nodes to hundreds of thousand nodes. And they can be static or mobile. In order to aggregate all the aforementioned characteristics, the node platform has some drawbacks, e.g., short communication range, low bandwidth, small memory, and limited battery. These drawbacks are restrictions that are inherent to any WSN application.

The feasibility of WSN is highly dependent on application requirements mainly due to the restrictions aforementioned. Even when the application has similar requirements to other previously deployed application, a complete application verification and validation must be performed again in the new environment. The main cause is that the behavior of a WSN can differ from the other previously known application, and the target environment can be different.

The development of WSN applications is challenging and demands a peculiar effort. The difficulties include requirements satisfaction, gap between model and implementation, specific hardware platform, system validation, verification, testing [1] [2]. Then, during the WSN application development the team members inform the project management about some concerns related to prototyping. For example, hardware test and approval, application test, and third-party system integration evaluation are related to prototyping. So, we argue that one should consider the use of specific software development process to improve the development organization, deal with such concerns, and to enable a better project management. When the organization uses a process for software development, it can have the opportunity to reproduce the process in the following projects and to enhance it with the feedback of the previous team. Software development process [3] is a set of activities whose goal is the development or evolution of software. An example of software development process is Open Unified Process (OpenUP) [4].

OpenUP is an open iterative software development process that is intended to be minimal, complete, and extensible [5]. It can be used to develop software of different purposes, from small and embedded to desktop enterprise application. For example, some WSNs projects are Aquila Tower Monitoring [6], and Aqua WSN [7]. Nevertheless, OpenUP does not address the specific requirements for development of WSN systems. Thus, it is difficult to achieve a predictable system behavior that complies with the WSN application requirements.

There are some studies reporting the extension and integration of OpenUP, such as [5] for specifying capacity requirements, and [8] for security. Yet complex products, such as WSN, need special handling of requirements as well. However, there is a lack of definition in the existing software development process to address the development of a WSN application project. The main reason why development

Thanks to CAPES process nr. 6804-14-4.

 ISBN: 978-1-61208-367-4

processes do not fit to WSN development is the inability to meet important requirements of WSN satisfactorily and they do not provide a way to manage activities that are needed through the development, such as, simulation and prototyping activities.

In addition, other characteristics that need to be addressed during the development process of the WSN applications are the data integration with third party systems, employment of verification techniques, and different viewpoints used for analysis. Yet, due to the development characteristics of WSN systems, it is important to aid the development team with a disciplined way to perform their tasks, including prototyping activities.

This paper is organized as follows. Section II investigates some related works. Section III introduces the Prototype Disciplines that can be integrated into software development process for WSN applications. Section IV presents details of work products integration from the Prototype discipline to the OpenUP development process. Section V illustrates the use of proposed discipline. Finally, Section VII concludes this paper.

II. Related Work

There are several studies reporting processes for software development in constrained environments [9] [10] [11] [12]. However, there is not a specific process to address the development of WSN applications. Developers should be aware of restrictions such as limited storage, battery consumption, low accuracy sensor, and short transmission range. WSN are highly dependent on application requirements, then one application is hardly equal to another.

Carvalho et al. [9] conducted a comparative investigation between applications of two software development processes: Scrum[13] and Rational Unified Process [14]. The authors concluded that it is necessary high effort in the traditional method compared to the agile software development.

According to Marincic et al. [10], when the next generation of a system is designed, the new system will have common elements with its previous version. Then, the authors propose a framework for identifying the non-formal elements of knowledge which can support modelling of the next system generation. The authors presented the application their framework modelling mechanical parts of a paper-inserting machine on an action research case.

The development process proposed by Nosseir et al. [11], called Mobile Development Process Spiral, is a usability driven model designed to integrate usability into existing application development processes. It also recommends usability techniques for assessing mobile applications. The proposed method aims identifying a set of usability techniques and incorporating these techniques into each iteration to assess the mobile applications.

According to Berrani [12] WSN specification is a complex task due to their embedded and distributed nature as well as the strong interaction between their hardware and software parts. In order to improve the verification of the WSN properties the authors propose an approach called Model Driven Architecture (MDA). This approach aims to promote the reusability and improve the development process. The authors mentioned that their approach promoted the reusability of modeled components and it also facilited the modeling task decreasing relative costs.

III. Prototyping Discipline

The use of prototypes during project development helps to improve the knowledge about the system, the network, and the nodes. In this research a prototype has at least one real node and the working code is deployable to the node's hardware. The scope of discipline is restricted to software prototypes. Then the prototype discipline can be used at any process phase of the OpenUP.

The prototyping discipline considers specific tasks to build a prototype during the software development process for WSN application. The workflow is depicted in Figure 1. The workflow begins by defining the objective for the prototype study, it drives all the further decisions. After that, a specification regarding the prototype requirements is elaborated. Afterwards, the prototype design based on information from previous activities performed can be started. Then, in parallel, the development and the calibration activities are performed, followed by the test activity, which verifies if the code complies with the objectives. So, the code is compiled and deployed in the nodes to perform the experiments which can begin. After running the experiments, the results are evaluated and depending on them it might be needed a code review and new experiments or it proceeds to document the generated results. In the remainder of this section details of each activity of this discipline are given.

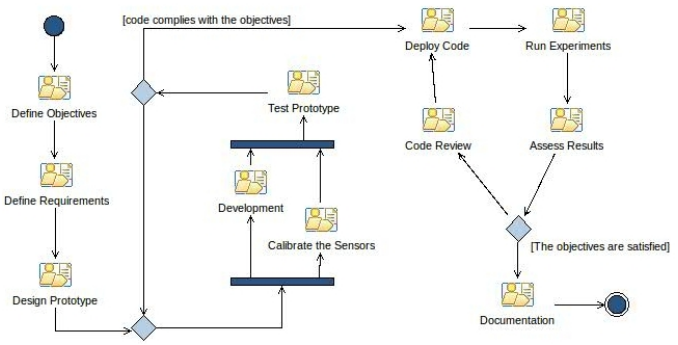


Figure 1. Workflow for prototype discipline

A. Define Objectives

This activity defines which is the objective for the prototype discipline. The prototyping model used to perform this discipline relies directly on the chosen objective. For example, one can create a prototype only to test the sensors and after the study has finished, the prototype is not needed any longer. This activity is performed by *Prototype Analyst* and the main output work product is the *Prototyping Objectives*. This activity has two tasks: *Define Objectives*, and *Plan Prototyping*. The first task aims to define the overall prototyping objective and the second one aims to plan the prototyping activities. The *Prototyping Plan* is used as an input work product of the design task performed later.

 ISBN: 978-1-61208-367-4

B. Define Requirements

After the objectives are defined, the definition of the prototype requirements is produced. The requirements drive the prototype development. It is performed by the *Prototype Analyst*. This activity produces the *Prototype Requirements Specification* work product.

C. Design Prototype

This activity consists of describing the blueprints that guide the development. Some information in this activity describes, for example, the detailed communication model, topology to be followed, which programming approach to be used and a detailed software architecture, and even the base station behavior if it is applied for the study. This activity is performed by the *Prototype Analyst* role and he can be assisted by the *Prototype Developer* role. The *Software Specification*, *Protocols Definition*, and *Base Station Behavior Description* are the main output work products of this activity.

D. Develop Prototype

In this activity, the code is implemented. This activity is performed by the *Prototype Developer* role and it produces the *Code* that is compiled and deployed to the real nodes. In addition, the *Prototype Developer* needs to deal with the WSN restrictions during the coding tasks. He/She also needs to deal with the specified middleware, OS, and hardware platform details. The role has to manage the code size, because of the memory size available in the hardware platform. It is expected that this activity demands a considerable effort. Finally, in order to assist the coding, one can use the simulation model for, if it is applied, developing the WSN application.

E. Calibrate Sensors

This activity is performed by the *Phenomenon Specialist* and he can be assisted by the *Prototype Developer*. It consists of verifying the values of the prototype and then calibrates it closer to the real values as possible. So, during the experiments the values obtained from the prototype are more reliable. This activity has the *Sensor Calibrated* as an outcome.

F. Test Prototype

Tests are performed in order to verify if the code complies with the prototyping objectives of the current iteration and if there is no error in the code. This activity is performed by the *Prototype Tester* and it is expected as results the *Test Log* work product. The *Test Log* contains the results of performed tests and if some test fails the process resumes back in the *Development* activity otherwise it proceeds to *Deploy the Code* activity.

G. Review Code

The *Review Code* activity is only performed if the experiment results do not fulfill the prototype objectives and requirements. It consists of performing a review of the code deployed after running the experiments and assess its results. The *Prototype Developer* does the solicited changes in *Code Review* task and then the *Prototype Tester* performs again the tests in the code reviewed. The process resumes in the *Deploy Code* activity only when the complete review is finished.

H. Deploy Code

Deploy the code in the nodes is an activity performed by the *Prototype Developer*. It is performed using the tools available from the chosen OS. In some cases, it can be made via over-the-air communication, but commonly the node is plugged to the development station and then the binary code is deployed to the connected node through the cable. The outcome from this activity is *Prototype Nodes with Code* loaded. The task *Deploy* is performed with the nodes that are considered for the experiment.

I. Run Experiments

This activity is executed by the *Prototype Developer* with the aid of the *Prototype Analyst*. *Run Experiments* activity aims to obtain the experiments results that are used in the *Assess the Results* activity in order to evaluate the current study. The main output work product is the *Experiment Results*.

J. Assess Results

After the experiments results are available, further analysis are performed. The *Prototype Analyst* role is responsible for performing the analysis and he can have assistance from the *Phenomenon Specialist* role. The analysis can be made using statistics techniques and tools. In addition, the activity provides information, via its main output work product *Experiment Results Evaluated*, to make a decision if the experiments results satisfy the prototype's objective. If the results do not fulfill the objective then the process resumes in the *Code Review* activity. Otherwise, the next activity is the *Documentation* activity.

K. Document Prototype

This activity consists of evaluating the results provided by the *Assess Results* activity. The *Prototype Report* is generated. In this report some information is available, such as solicitation for requirements, parameter, and sensors review, or it can provide information how close the prototype is to reach the quality of the final system. The report can have also decisions resulting from the experiments evaluated regarding the prototyping objectives. The *Prototype Analyst* role performs the *Documentation* task.

IV. Integration with OpenUP

During the development process, the team members inform the project management some concerns that include: further requirement analysis, debugging results, communication analysis, system performance analysis, design evaluation, scalability validation, hardware test and approval, application test, third-party system integration evaluation, requirements refinement, and training results. Some concerns of hardware test and approval, application test, third-party system integration evaluation, requirements refinement are related to prototyping. The concerns are reported to project management and then a decision regarding of which approach is used in sequence is taken.

The project team performs an assessment of the raised concerns and then the related information is updated to the following work products: *Iteration Plan*, *Project Plan*, *System Wide Requirements*. These work products are the main inputs

 ISBN: 978-1-61208-367-4

to start the prototyping iteration. After the selected iteration is executed, it provides a report with the answers and recommendations to the project. The prototyping discipline provides the *Prototype Report*. It is important to mention that this report is not the only work product outputs, the iteration also generates other work products of interest, for example, a validated model, experiments results assessed, and application code evaluated.

Table I presents details of work products integration from the Prototype discipline to the OpenUP development process. Table I also describes how the work product is used by the proposed disciplines: as input or as output. It indicates which tasks use the work product as input and which tasks produce the work product.

TABLE I. Work products interaction of Prototyping Discipline

Work Product (WP)	**Type**	**Task that produces the WP**	**Task that uses the WP**
Iteration Plan	Input	*Plan Iteration/ Manage Iteration*	*Define Objectives*
Project Plan	Input	*Plan Project*	*Define Objectives/ Plan Prototyping*
Vision	Input	*Develop Technical Vision*	*Requirements Specification/ Define Objectives*
System-Wide Requirements	Input	*Identify and Outline Requirements/ Detail System-Wide Requirements*	*Requirements Specification*
Middleware Definition	Output	*Design Prototype*	*Implement Solution*
Protocols Definition	Output	*Design Prototype*	*Implement Solution*
Message Behavior Description	Output	*Design Prototype*	*Implement Solution*
Software Specification	Output	*Design Prototype*	*Implement Solution*
Base Station Behavior Description	Output	*Design Prototype*	*Implement Solution*
Code	Output	*Prototype Building*	*Implement Solution*
Test Case	Input	*Create Test Cases*	*Test*
Prototype Report	Output	*Documentation*	*Assess Results/ Detail System Requirements*

The following comments are related to the tasks that use the work products of the Table I. The *Define Objectives*, *Plan Prototyping*, and *Requirements Specification* tasks use the *Vision*, *Iteration Plan*, *Project Plan*, and *System-wide Requirements* work products that provide essential information to perform the tasks. The *Implement Solution* task is performed after the *Inception* phase is executed. It uses the following work products: *Middleware Definition*, *Protocols Definition*, *Message Behavior Description*, *Software Specification*, *Base Station Behavior Description*, and *Code*. The next work product, *Test Case*, is used in *Test* task. The input work product is the *Test Case* - it has the specification of a set of test inputs, execution conditions, and expected results related to some scenario. The work product *Prototype Report* is the most useful to the development process because it provides a feedback of the prototype iteration to the *Assess Results* task and the *Detail System-Wide Requirements* task. For example, the *Prototype Report* contents are details about the network protocol performance, and if the employed network protocol complies with the application requirements.

Therefore, if the integration requirements are satisfied, e.g., the process analyst can map all input and output work products, then our proposed extension could be integrated to other software development process. Thus, it can be used for development of WSN applications as needed.

A. Publishing

In this paper, Prototyping Discipline is described using the Eclipse Process Framework (EPF) – version 1.5.1. EPF enables the process manager/analyst to update all components of the process in use and also enables to publish the desired process. The process can be available directly inside the tool or it can be published for web access. The availability inside the EPF helps the process analyst to preview the web site structure before publishing it. For authoring, the EPF has available a perspective which provides the necessary set of solutions for method composing.

If the organization needs to provide access via web, then the process analyst just needs to generate the web pages, and then they can be published, for example, in the intranet of the organization or in the public web site. So, the published web pages are available via the web browser. For example, Figures 2 and 3 depict our proposed extensions available using the web browser.

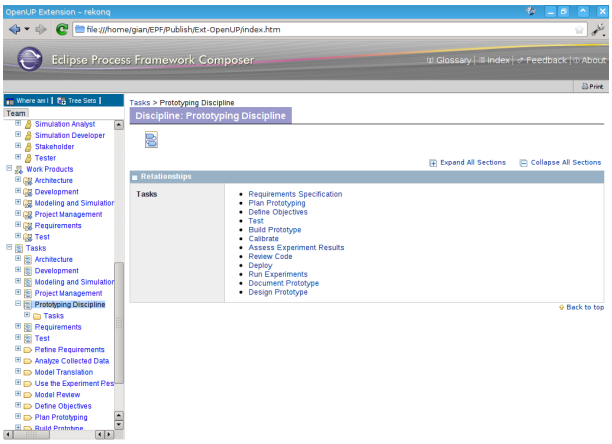


Figure 2. Consulting the extension via web browser

V. CASE STUDY

This section aims to illustrate the use of prototype discipline for the software development process for WSN application. The case study is about monitoring a cellar used to age Brazilian sugar cane spirits. A description of the application, some requirements, and the achieved results are given.

A. Description

The sugar cane spirits (Cachaça) is a genuine Brazilian drink, known worldwide. Its production began in the sixteenth century. According to Brazilian laws, which standardize and rank drinks, cachaça is defined as a typical beverage produced

 ISBN: 978-1-61208-367-4

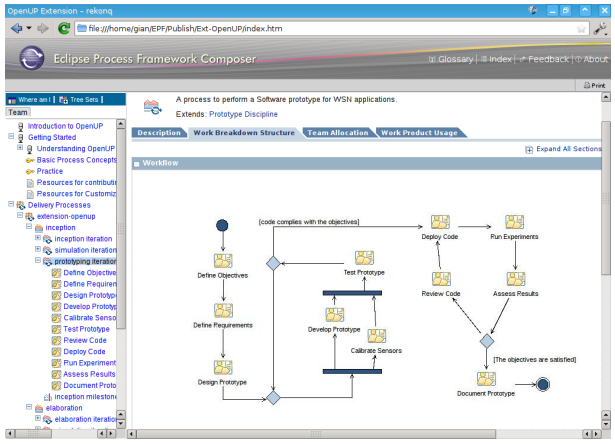


Figure 3. Consulting the prototyping workflow via web browser

in Brazil, with an alcoholic strength of 38 % vol (thirty-eight percent by volume) to 48 % vol (forty-eight percent by volume) at a temperature of 20^o C (twenty degrees Celsius). It is obtained by distilling the fermented juice of sugar cane with peculiar sensory features.

The production cycle of sugar cane spirits starts with the milling of sugar cane, through the preparation of the wort, fermentation, distillation, filtration and dilution. After filtering and resting, the sugar cane spirits can be bottled or stored in wooden barrels for aging.

The longer the aging of the sugar cane spirits, the higher is the value of the drink. In the aging process, the characteristics of the sugar cane spirits change, improving their qualities with new flavors, new tastes and new coloring.

B. User Environment

The main environment place is a cellar for aging drinks. The tasks related with the application in the environment are: deployment task, monitoring task, maintenance task. The considered platform is TelosB. TelosB is a WSN platform with a TI-MSP430 micro controller, 128 kbytes of memory, supported by TinyOS and ContikiOS, and it has a light, temperature, and humidity sensor. The maintenance task occurs only once a year. The monitoring task must inform the control station when the cellar is out of its environmental specification. So, in order to keep the cellar with the appropriate condition, a notification must be delivered. The previous notification occurs when the cellar environment reaches a critical temperature or humidity, close to the upper or lower bounds. Another notification must be sent if the cellar is out of its upper or lower bound for any given environment condition. The control station must store all the received events.

C. Functional Requirements

The sugar cane spirits age in a cellar with dimension of 8 meters wide per 80 meters long and 5 meters high. The cellar can store 800 barrels with capacity of 250 liters each or 200,000 liters in total. The barrel dimensions are 95 cm of height, 72 cm of diameter at middle, and 58 cm of diameter at top/bottom. During the aging time (from 1 year up to 5 years) the temperature must be within the range of 15^o C (fifteen degrees Celsius) to 20^o C (twenty degrees Celsius) and the air relative humidity must be from 70% to 90%. Figure 4 shows the arrangement of barrels in the cellar and the circles represent the nodes' positions.

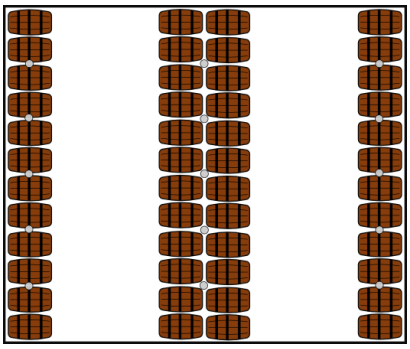


Figure 4. Cellar illustration with the barrels and the position of the nodes

In WSN application, the nodes must be able to set upper and lower bounds for the environmental variables (temperature and humidity). If the sensed value gets closer to the bounds, the node must send a message to the control station. It only stops sending the message when the control station notifies that the message was received. The node sends the message again at a 5 time intervals. Nodes should be able to connect to the network and it must be able to perform the following activities: sensing, routing, disseminating, aggregating. A threshold should be in two levels, either for upper and lower bounds, one soft threshold 10% lower than the hard threshold. All the parameters must be changeable and be retrievable through the control station. At least, once a day, the node must inform its sensing values to the sink node. All information received in the sink node is retransmitted it to the control station. When all nodes send the alarm message to the control station, they indicate their battery levels. The nodes are deployed one meter above the barrels and two meters from each other in the same row.

The Control Station communicates to the network via the node sink, i.e., it is directly connected to the sink node. It must also indicate the battery level of all nodes that perform any communication with it. Additionally, it must indicate the upper and lower threshold used by each node. The Control Station must be able to set a parameter of a single node or the whole network, and it must indicate all alarms received and notify back to the source node.

VI. Results

The case study was performed by only one person. He performed all the roles. During the execution of one simulation study, the time needed to execute each activity was measured. The whole simulation discipline was completed in fourteen days, including weekends and the prototype discipline was performed in 21 days.

Figure 5 depictes the relative time spent for each activity. The activities with zero time were not executed. And before, after, and between the execution of the proposed discipline, an Inception iteration from OpenUP is executed. Then, the iteration executions are Inception, Simulation, Inception, Prototyping, and Inception.

 ISBN: 978-1-61208-367-4

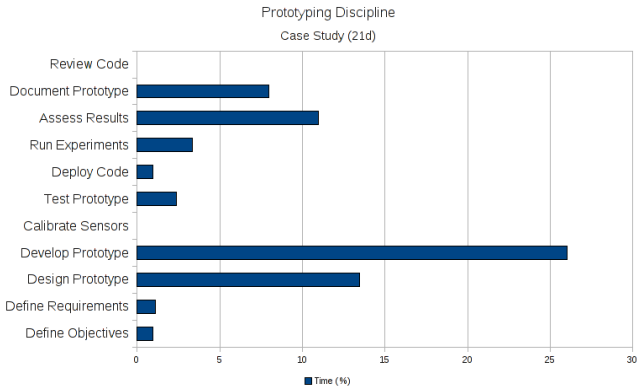


Figure 5. Relative time to perform each activity

After the entire process execution, the results are compiled and they show as expected. In each experiment, we changed different MAC protocol parameters, such as, duty cycle, listening interval, and transmission power.

Furthermore, the activities that demanded more effort to complete in the prototype discipline was the *Development* followed by *Design Prototype* and then by *Assess Results*. The longest activity is *Run Experiments* that took more than eight days to complete.

Using this discipline to perform the prototyping aided the developer to organize the work and to be more objective in each step of the process.

VII. Conclusion

This paper presented an extension for OpenUP. The purpose of this paper is to introduce the discipline Prototyping for developing WSN applications. The prototyping discipline was created in order to improve the knowledge about the system as well as to refine information about the system. This paper detailed the discipline and described the discipline activities. In addition, there is a description about how this discipline integrate with OpenUP and we argue that if the process analyst maps all input and output work products in a new software development process then the discipline can be used in that software development process. The EPF publishes the discipline by making available via web.

References

[1] A. Hasler, I. Talzi, J. Beutel, C. Tschudin, and S. Gruber, "Wireless sensor networks in permafrost research: Concept, requirements, implementation, and challenges," in Proceedings... International Conference on Permafrost (NICOP), 2008, pp. 669–674.

[2] A. Lédeczi, P. Völgyesi, M. Maróti, G. Simon, G. Balogh, A. Nádas, B. Kusy, S. Dóra, and G. Pap, "Multiple simultaneous acoustic source localization in urban terrain," in Proceedings..., International Symposium on Information Processing in Sensor Networks. Piscataway, NJ, USA: IEEE Press, 2005, p. 69.

[3] I. Sommerville, Software Engineering, 7th ed. Addison-Wesley, 2004.

[4] IBM, "Openup," 2009.

[5] A. Borg, K. Sandahl, and M. Patel, "Extending the openup/basic requirements discipline to specify capacity requirements," in Proceedings..., IEEE International Conference on Requirements Engineering. Los Alamitos, CA, USA: IEEE Computer Society, 2007, pp. 328–333.

[6] M. Ceriotti, L. Mottola, G. P. Picco, A. L. Murphy, S. Guna, M. Corra, M. Pozzi, D. Zonta, and P. Zanon, "Monitoring heritage buildings with wireless sensor networks: The torre aquila deployment," in Proceedings..., International Conference on Information Processing in Sensor Networks. Washington, DC, USA: IEEE Computer Society, 2009, pp. 277–288.

[7] H. Ntareme, M. Zarifi, A. Ergawy, S. Rathore, K. Wang, and A. Strikos, "Aqua wireless sensor networks," January 2008. [Online]. Available: http://www.online.kth.se/csd/projects/0726/

[8] S. Ardi and N. Shahmehri, "Integrating a security plug-in with the openup/basic development process," in Proceedings..., International Conference on Availability, Reliability and Security. Washington, DC, USA: IEEE Computer Society, 2008, pp. 284–291. [Online]. Available: http://portal.acm.org/citation.cfm?id=1371602.1371921

[9] S. Carvalho, F. Motta Cardoso, A. Da Cunha, and L. Zanetti, "A comparative research between scrum and rup using real time embedded software development," in Information Technology: New Generations (ITNG), 2013 Tenth International Conference on, April 2013, pp. 734–735.

[10] J. Marincic, A. Mader, R. Wieringa, and Y. Lucas, "Reusing knowledge in embedded systems modelling," Expert Systems, vol. 30, no. 3, 2013, pp. 185–199. [Online]. Available: http://dx.doi.org/10.1111/j.1468-0394.2012.00631.x

[11] A. Nosseir, D. Flood, R. Harrison, and O. Ibrahim, "Mobile development process spiral," in Computer Engineering Systems (ICCES), 2012 Seventh International Conference on, Nov 2012, pp. 281–286.

[12] S. Berrani, A. Hammad, and H. Mountassir, "Mapping sysml to modelica to validate wireless sensor networks non-functional requirements," in Programming and Systems (ISPS), 2013 11th International Symposium on, April 2013, pp. 177–186.

[13] K. Beck and C. Andres, Extreme programming explained. Embrace change. Addison–Wesley Boston, 2005, vol. 2.

[14] P. Kruchten, The Rational Unified Process: An Introduction, 3rd ed. Addison Wesley, 2003.

 ISBN: 978-1-61208-367-4

Easily Evolving Software Using Normalized System Theory A Case Study

Gilles Oorts, Kamiel Ahmadpour, Herwig Mannaert and Jan Verelst
Normalized Systems Institute (NSI)
University of Antwerp
Antwerp, Belgium
{gilles.oorts,kamiel.ahmadpour,herwig.mannaert,jan.verelst}@uantwerp.be

Arco Oost
Normalized Systems eXpanders factory (NSX)
Antwerp, Belgium
{arco.oost}@nsx.normalizedsystems.org

***Abstract*—Software agility is characterized by inevitable software changes and ever-increasing software complexity. Unless change accommodations are rigorously taken into account, the implementation of these changes may lead to exorbitant costs. This is in particular true for long-lived systems. For such systems, there is a need to explicitly address evolvability concerns during their design phase. This to carry out software evolution efficiently and reliably during their lifecycle, and prolong the productive life of the software systems. Normalized Systems (NS) theory has been recently proposed as an approach to develop agile and evolvable software. In this paper we discuss the practical advantages of the NS approach using a case study regarding the revision of a budget management application. Furthermore, advantages such as knowledge transfer through the NS development process are also discussed in this paper.**

Keywords–Normalized Systems theory; Evolvable Software; Adaptive Software; Agile Software; Case Study

I. INTRODUCTION

In ever-increasing volatile environments, evolvability is considered as one of the most important characteristics of information systems. As information systems support the operations and decision-making of organizations, software applications also need to support the changes on the business-side of organizations. However, organizations normally find it difficult to synchronize changing requirements needs with their software applications. This is because the current software development paradigms do not fully take into account the changeability of business needs over the life cycle of software systems. This problem is also characterized by the Law of Increasing Complexity as proposed by Lehman, which states that the structure of software tends to become more and more complex over time because of changes made to the software [1]. Furthermore, software applications are traditionally built to last several years -or even decades- in order to justify (high) development costs. With regard to changing business requirements, this often leads to decisions to either not implement the changes because they are too expensive, or to eventually (after several years or decades) totally scrap the application and start the development of a new "up-to-date" application.

Recently, Normalized Systems (NS) theory has been proposed as a way to deal with ever-changing requirements for software by building evolvable information systems, based on the systems theoretic concept of stability [2]. As recent research shows, these systems are capable of incorporating changes more easily and with less effort by means of a careful design of the software architecture [3], [4]. Therefore, changes can be made immediately and the life cycle of software applications is greatly extended, up to a point that they can be used and revised infinitely. In this paper we will discuss how changing business requirements can be easily implemented into an application developed according to the NS theory. This will be illustrated by means of a case regarding the revision of a budget management application. The initial development of this budget application is described in [4], which focused on illustrating the NS development methodology used in developing the application. In this paper we provide a clear understanding of the NS advantages in dealing with changes to this initial application by comparing both versions of the software, the scope of changes and the amount of time and effort spend on implementing all the updates in the new version. As the case description requires an understanding of the NS theory, a brief review of the NS theory is provided in Section II. For a more thorough description of the NS theory, we refer to a wide number of previous works (for example, [2], [3], [5]). In Section III, we will first provide a short description of the initial software application, followed by the changed business requirements. How these changes were implemented according to the NS theory is discussed in Section IV. In Section V, we will discuss the advantages of NS development, some observations, and contributions of the paper. We end the paper with a brief conclusion regarding this paper in Section VI.

II. NORMALIZED SYSTEMS THEORY

NS theory is concerned with how information systems can be deterministically designed and developed based on the systems theoretic concept of stability. According to NS, the main obstacle to evolvability is the existence of so-called combinatorial effects. In this condition, the amount of effort to make a specific change in the system is not only related to the change but also to the size of the system. Therefore the effort to apply a specific change increases as the system grows [2]. According to the systems theory, stability refers to a system in which a bounded input function results in bounded output values, even as $t \rightarrow \infty$ (with t representing time). When applied to information systems, this means that applying a specific change to the information system should always require the same effort [3]. According to NS theory, the avoidance of all combinatorial effects in software leads to evolvable software, as this means ripple effects of changes do not increase over time and, as such, with the size of the system. To eliminate combinatorial effects, NS theory proposes a set of four theorems and five elements that can be expanded into

 ISBN: 978-1-61208-367-4

fully functional applications through pattern expansion. This set of theorems and elements are the foundation of NS theory, and will be discussed in the next sections.

A. NS Theorems

- *Separation of Concerns (SoC)*, requiring that each change driver (concern) has to be separated from other concerns. This theorem allows us to isolate the impact of each change in its own entity;
- *Data Version Transparency (DVT)*, requiring that data entities can be modified (e.g., additional data can be sent between components), without having an impact on other entities;
- *Action Version Transparency (AVT)*, refers to a condition in which an action entity can be upgraded without impacting the calling components;
- *Separation of States (SoS)*, implies that actions or steps in a workflow are separated from each other in time by keeping a state after every action or step.

These are just brief definitions of the NS theorems, as these have previously been extensively discussed in other work (e.g., [2], [3], [5]). It has to be mentioned that none of these theorems are completely new, and even relate to heuristic knowledge of developers [5], [6]. However, formulating this knowledge as theorems aimed at identifying combinatorial effects will help to build information systems that contain a minimal number of combinatorial effects. Only when the design of an application completely adheres to the NS theorems, one can profit from the software evolvabilty that the NS theory offers.

B. Normalized Systems Elements

As the systematic application of the NS theorems results in a very fine-grained modular structure, NS theory proposes to build information systems based on the aggregation of instantiations of five higher-level software patterns or elements, being:

- *a data element*, representing an encapsulated data construct with its get- and set-methodss to provide access to its information in a data version transparent way. Cross-cutting concerns (for instance access control and persistency) are considered to be a part of the data element;
- *an action element*, containing a core action representing a single change driver or functional task;
- *a workflow element*, containing the sequence in which a number of action elements should be executed in order to fulfill a flow;
- *a trigger element*, controlling the states (both regular and error states) and checking whether an action element has to be triggered accordingly;
- *a connector element*, ensuring external systems are able to interact with the NS system without allowing elements to be called in a stateless way.

The above mentioned NS elements are the essential building blocks for a NS application and provide the core functionality of an information system. They can then be easily extended later (cf., description of extensions in Section IV). A functional analyst will formulate instantiations of the NS elements that are the foundations of a NS application [4]. At run time, these instances are instantiated once more (i.e., constitute a double instantiation) to form specific occurrences of, for example, a budget [4].

The NS elements have been described more extensively in [2], [3], [5] and the implementation of a data element in a Java Enterprise Edition (JEE) has been described in a previous work [5]. The definition and identification of the NS elements is based on the implications of the set of NS theorems [7]. For example, the definition of the workflow element is based on the Separation of Concerns (SoC) and Separation of States (SoS) theorems. In a workflow element, we can invoke action elements in a completely stateful manner and as mentioned earlier, keeping track of every action's state, leads to Separation of States (SoS). Similarly, each of the five NS elements constitutes one possible solution for implementing all four NS theorems, thus eliminating all combinatorial effects.

Each of these five elements provides a general reusable solution to a commonly occurring problem within a given context. Therefore, they can be considered as a design pattern, containing a core construct and several cross-cutting concerns (such as remote access, logging, access control, etc.). This architecture provides protection from combinatorial effects while allowing for a set of anticipated changes to be applied to a system [5]. As such, the five NS elements can be used to build an evolvable information system that satisfy the four NS theorems.

C. NS Pattern Expansion

The use of NS elements as design patterns is supported by the NS pattern expansion mechanism, which enables the conversion of NS element instances defined by the developer into fully functional code. Without using such expanders, it would be near to impossible to achieve the fined-grained modular structure prescribed by the NS theorems. Therefore the NS expanders are considered to be an essential part of designing an application using NS theory.

As such, pattern expansion is one of the four phases in the NS development process [4]. First a comprehensive functional analysis is performed to identify the NS element instances. Coding these instantiations is the next step of this process and it is done using some special "descriptor files". A descriptor file is a text or XML-based file which constitutes the input for the NS expanders.

For example, in case of a data element instance, one needs to provide the following parameters in the descriptor file in order to be able to work with the expanders:

- *Basic name of the data element instance:* Each data element instance needs to have a unique name which needs to be provided in the descriptor file (e.g., Budget).
- *Context information:* It provides the package and component name of the data element instance.
- *Data field information:* Each data element instance can contain one or more data fields and all the information about these data fields (such as their name and data type) needs to be provided in the descriptor file;
- *Relationships with other element instances:* It is necessary to address all the relationships of current data element instances with other element instances.

 ISBN: 978-1-61208-367-4

In the next step, through the process of expansion, the descriptor files get expanded into the code of a functional application. This show how a minimum of input information in the descriptor files can be used to transform into a fully functional application. The NS pattern expansion is done by software (called NS expanders) developed especially for this purpose by the NS eXpanders factory (NSX).

The NS expanders expand the descriptor files into skeleton source code for all the identified NS element instantiations. Furthermore, the NS expanders also provide all deployment and configuration files required to construct a working application on a supported technology stack. The skeleton source code facilitates a top-down design approach, where a functional system with complete high-level structures is designed and coded, and this system is then progressively expanded to fulfill the requirements of the project. These expansions are called NS extensions and will be discussed later in this paper. The classes of the skeleton code represent the modular structure of the defined NS element. For the Budget element instance, the NS expanders will for example generate a set of classes and data fields such as the bean class BudgetBean and its related local and remote interfaces (BudgetLocal and BudgetRemote).

Because of the NS expansion mechanism, applying changes to the application only requires us to provide new descriptor files and a re-expansion of these updated files will provide a new version of the application. This process will be shown in the section discussing the implementation of the changes to the application presented in the next paragraphs.

III. The NS Budget Management Application Case

Over the last years, several applications have been built based on NS theory and its development methodology. The most extensive description of this methodology can be found in [4], in which it is discussed by using the development of a budget management application for a local Belgian government as an exemplar. After using this application for only one year, both regulatory and requirements changes required the application to be changed. As the initial application was built according to the NS principles discussed in the previous section, these changes could be implemented rather easily. In the next paragraphs, we will first explain the functionality and design of the initial application, followed by an overview of the change requests.

A. The Initial Budget Management Application

Budget tracking and management are important aspects within the administration of the local government. Budgets need to be awarded, specified, managed and utilized for the local government to function properly and fulfill its services to the citizens. To accomplish this, the overall available budget is divided into very fine-grained sub-budgets. This however drastically complicates the budget assignment, reservation, fixations, changes, etc. To cope with this complexity -and simultaneously realizing the much-needed integration of budget management with project management and budget reporting- a project was started at the end of 2012 to develop a stand-alone application to capture the budget management functionalities.

The challenges of the budget application development are discussed in [4]. First, the new application needed to match the flexibly and user-friendliness of Excel pivot tables (which were previously used). To cope with this challenge, the development of the initial application was focused on budget management functionalities and its user-friendliness. As will be discussed in this paper, the initial application would, as such, be a sound basis for further extending the application to include other requirements such as budget reporting, simulation and project management, etc.

The context-specific and fine-grained composition of the budgets was another challenge of the application development. Budgets need to be defined in a range of different levels of the specific government. Therefore budgets can be managed on a scale from general to highly specific. On the most fine-grained or specific level, budgets are defined by a combination of the following six parameters: department, activity, article, domain, product and budget year. However, budgets can also be defined based on a subset of these parameters, meaning budgets can be defined on several levels.

After going through a functional analysis and NS software development process, the final application architecture is shown by the set of unchanged, changed and removed data element instances in Fig. 1. This figure clearly shows how the application is structured around a central data element instance, being a “Budget”. A current budget is defined by the aggregation of “Budget Changes” made to the budget over time. The parameters that can be used to define a budget - “Department”, “Activity”, “Article”, “Domain”, “Product” and “Budget year” instances- are shown on the left side of Fig. 1. These “Articles” can be grouped into “Economic groups”. In the initial application, Economic groups make up a “Budget estimate”. This estimate used to be utilized to draw up a target budget at the beginning of a budget year. The data elements instances on the right of the Budget instance in Fig. 1 are used for managing budgets. “Budget fixations” are used secure a part of a budget for a specific purpose. These Budget fixations are assigned to a specific “Supplier” and can be called in “Budget calls”, so the budgets can be partially spent when needed. Budget calls are associated with “Invoices” and “Work orders” to track the spending of “Budget calls”. Other aspects of the application will be discussed in the next section, as they are part of the changes made to the initial application.

B. Change Requests for the Application

As mentioned before, the government officials had several change requests after having used the budget management applications for some time. Additionally, changes in legislation required the introduction of a “Purchase file” data instance to enable long-term (i.e., more than one year) tracking of purchases of departments. Thereby, this case show how an application that was in use for only one year already needed functional changes. This meant that a lot of software features had to either be added, changed or removed. One developer even expressed that one could argue the change requests were so extensive and concerned the foundation of the application that you could consider it as a new application. NS theory however allows for far-reaching changes to be made to an application, and the renewed application is therefore considered a second version of the budget application. In the following paragraphs, we will briefly review some of the changes that

 ISBN: 978-1-61208-367-4

have been made to the application, for which Fig. 1 can be used as a reference.

The first significant change was the addition of a project management functionality to the application. Because the NS methodology was used, this functionality could be left out in the initial application to be easily added later on. The project management functionality that needed to be added involved project monitoring through observation of the state of work orders. These work orders get initiated in the system for specific tasks and are linked to the budget fixation they belong to. When an invoice is received for a work order, the invoice lines are linked to specific work order lines. To fully implement the new project management functionality, new data element instances for "Consultant" and "Profile" needed to be added as well.

Another change request was the extension of invoice management. Invoices were made more detailed by adding a data element instance "Invoice line" and by linking invoices directly to the budget fixation they belong to. Previously invoices were defined on three levels: for fixations, budget calls and work orders. For simplification and centralization reasons, this was reduced to only one level in the revised application.

The third major change in the new application is the inclusion of purchase orders. These orders needed to be added to the application as the purchasing department needed to be able to control purchases according to the granted budgets. Additionally, a element instance for "Purchase File" was added to hold all the information on specific purchase orders.

Furthermore, it was made possible to further specify an activity by adding the data element instances "Action plan", "Policy domain" and "Management domain". And as budget estimates were not used in the application, the corresponding element instance was removed.

These changes show that only the fundamental functionality of the application stayed unchanged in the new version (i.e., the budget, budget change and budget-defining data element instances). With the exception of four element instances, all initial element instances needed to be "touched" and several element instances needed to be added to provide the requested functionality changes.

IV. Implementation of the Changes

As the goal of NS theory is to design software in an evolvable way, it should be no surprise that the change requests discussed in the previous section could be implemented quickly and easily by just a single developer. By taking evolvability of software into account at design time, NS applications can effortlessly be extended through the descriptor files and the expansion mechanism.

In the descriptor files of the NS element instances, one can for instance easily change the data model of an application (i.e., the relationships between element instances). For example, although the "Work order" and "Invoice" retained the same name in the second version of the budget application, their definition and position in the data model changed completely (cf. previous section). The changes could however be applied by simply re-defining the relationships of these element instances and the instances they are linked with in their descriptor files.

Similarly, adding or removing NS element instances from an application can also be done by just writing or removing descriptor files for these instances. The "Work order Line", "Invoice Line" and "Consultant" element instances for example could be added to the application by creating new descriptor files containing information such as their description, relationships, etc. Additionally, relationships to these new instances need to be added in existing element instances that are coupled with the new element instances. As these changes can be done rather easily, implementing all the required changes to the descriptor files of the budget application took less than 1 man-day.

Although the NS expansion process delivers a fully working application that includes all defined NS element instances, the functionalities of the application most likely still need to be extended to provide context-specific functionality. This is done through manually programming customizations, either within anchor points in the expanded files (called "injections") or in separate files (called "extensions"). The additional functionality added to the budget application needed two important extensions and/or injections.

As the budget application is very data-intensive (i.e., it exists of only NS data element instances), a lot of data validations needed to be implemented in the application. For example, budget calls can not exceed the available budget, budgets need to be unique, etc. Implementing these validations took 3 man-days.

The second important type of extensions that needed to be implemented were graphical extensions. These included colored status boxes, projections and HTML screens to implement these projections. Projections are views that can be defined on NS data element instances that show data in a specific way. These projections need to be defined so information can be projected to the end user in the most useful way in specific use cases. For example, when a user is looking up information on a specific budget, this screen also needs to display overall information on all budget calls made on this budget. From this overall information, the user can then select a budget call to get more information on. It however does not need to show detailed information on all budget calls in the budget screen, as this would lead to an information overload on this screen. Furthermore, different projections can be defined depending on user roles. For the 25 end users of the budget application, several roles are defined in the application (e.g., super user, administrator, manager, employee). As such, managers or members of a specific department can be shown more detailed information than regular employees. Implementing the custom graphical elements for the second version of the application took 5 man-days in total, of which the projections took 3 days to implement.

V. Discussion

From the description of the case in the previous section, one can make several observations.

First off all, the benefit of reduced effort of changing NS applications seems to come at an extra cost or effort at design time of the application. However, the additional effort to design in an evolvable way is negligible. Previous research has shown that the NS expansion process is very efficient

 ISBN: 978-1-61208-367-4

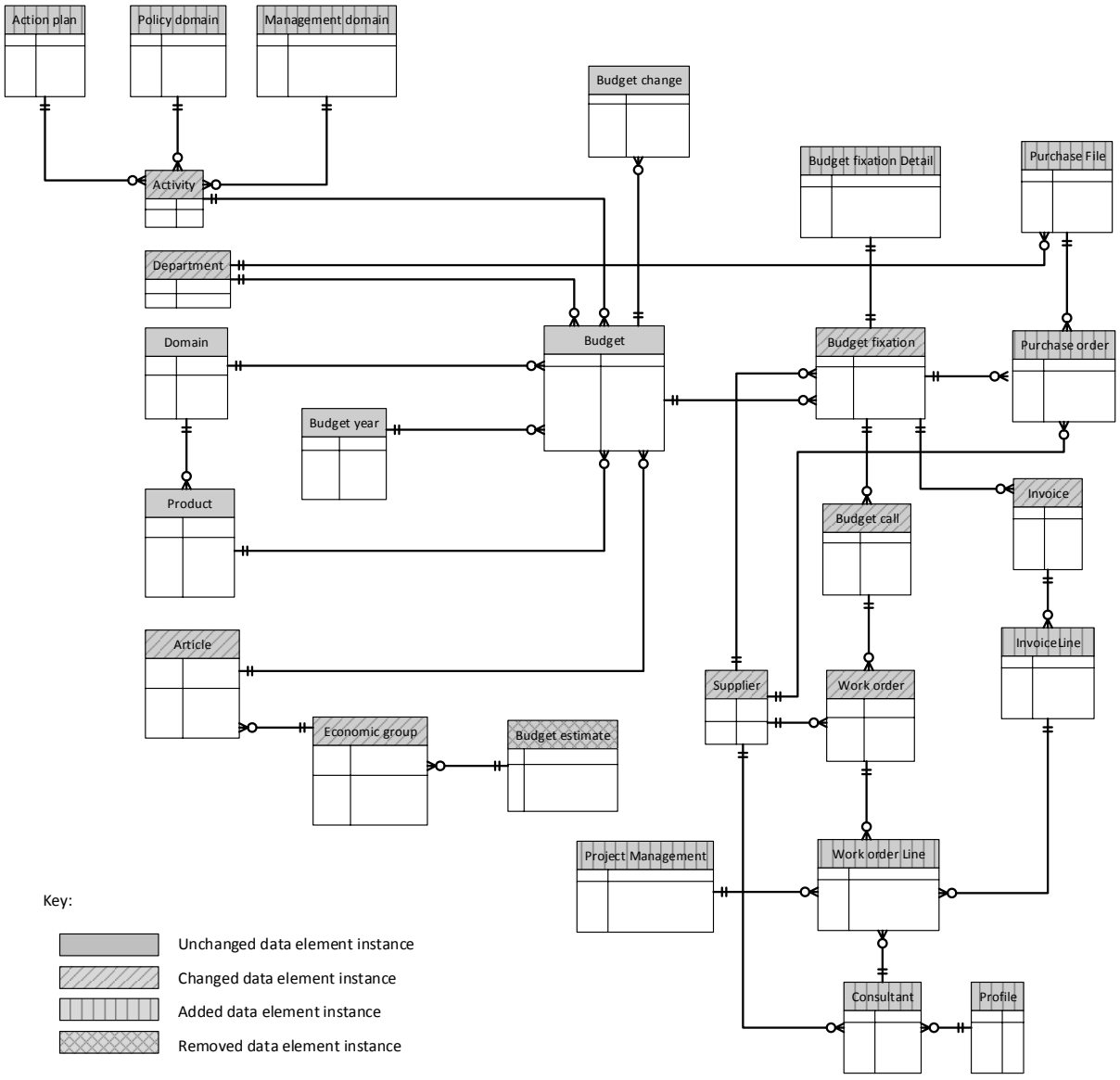


Figure 1. Entity Relationship Diagram Showing the Architecture and Changes to the Budget Application

and fast and even provides a way of developing software faster than traditional development methodologies [4]. This is because the developer does not need to concern himself with the software architecture or boilerplate code once the NS element instances are defined in descriptor files. The only prerequisite is that additional knowledge on the NS theorems, elements and expanders is required for developers to be able to develop software that is fully according to NS theory.

Once an application has been built according to NS principles, the case description also shows it can be easily changed. The total development time of the thorough changes to the budget application was only about 9 man-days. According to the developer, the entire job was very clear to him. And he reckoned the amount of effort he had to spent on re-developing the application was much less than something they normally do when an application is not designed based on NS theory.

A third observation is that because the rapid development of the new versions of an application, issues that are otherwise proportionally irrelevant, can become even more time-consuming than the development itself. For the revision of the budget application, there was a lot of effort needed to convert and input the old Excel-data in the new application. This is because of missing data, inconsistencies, wrong data formats, etc. Overall this even took more effort than developing the new version of the application.

A. *Knowledge transfer*

NS development also incorporates knowledge management processes that support capturing, storing, transferring and applying development knowledge. How this works is discussed in [7], based on the widely used theoretical framework of [8]. Basically, knowledge is captured and transferred through

the use of NS expansion, as this process provides a way to incorporate new insights obtained from practical application of the theory into the NS knowledge base (i.e., the NS expanders). Captured knowledge can be newly normalized features in the NS elements that can be re-used in future applications, new general reflections on building Normalized software, etc. This way, newly normalized features can simply be provided to new applications through NS expansion and they do not need to be manually added after expansion. As such, any new version of the NS expanders can as well be used to re-expand older applications to provide additional functionality, graphical and more user-friendly enhancements, etc.

During the development of the revised budget application, several insights gained from previous NS projects could be used, including the initial development of the budget management application. One example of functionality that could be added more easily in the application revision are composite screens. These advanced screens provide an overview of data on different levels. They show for example the budget of a department, and by selecting an activity, domain, article, etc., one can drill down to a specific budget on the same screen. Before the start of the initial budget application, implementing such screen would take about 600 lines of code in manually programmed extensions. By incorporating some of the functionality of composite screens in the NS expanders, this was reduced to about 60 lines of manual code during the development of the initial budget application. In this revision of the application the effort needed was even further reduced to about 5 to 10 lines of code for each layer in a composite screen. The development of the revised budget application also lead to the addition of new knowledge to the NS expanders. The idea of projections (cf., previous section) that only show relevant information on a NS data element instance (e.g., when in a list of departments, one is only interested in total budget of the departments) is very useful in a large array of contexts and applications. Therefore they were added to the expanders after completion of the project.

B. Contributions and future research

This paper has several *contributions*. First, it shows the advantages of building software according to the NS design theory. These advantages can normally only be observed over long periods of time, when systems are required to evolve or be adapted. The case description in this paper however already shows for the first time how fundamental changes can be made to an application without excessive implementation effort (e.g., implemented by only one developer over a very limited amount of development days). Furthermore, the absence of combinatorial effects in NS applications will also make sure that the effort of implementing changes does not increase over time, as the application becomes larger and more complex. Second, the case description shows how the NS development process (discussed in detail in [4]) also supports the implementation of changes to an application by using the descriptor files and NS expanders.

Possibilities for *future research* include additional case studies to provide more information on how the NS theory realizes profound progress with regard to the evolvabilty of software.

VI. Conclusion

In this paper, we discussed how software can be easily revised and adapted when it is built according to NS theory. This was shown by means of describing the revision of a budget management application built for a local Belgian government, of which the NS development is previously discussed [4]. This case shows how, even after only being used for a single year, changes needed to be made to the software design because of regulatory and requirement changes. As such, the paper shows how these fundamental changes can be made easily and without much effort. Because the case application was built according to NS principles, one will also be able to implement future changes with the same ease and the application will thereby become evolvable.

References

[1] M. Lehman and J. Ramil, "Rules and tools for software evolution planning and management," *Annals of Software Engineering*, vol. 11, no. 1, pp. 15–44, 2001.

[2] H. Mannaert and J. Verelst, *Normalized Systems: Re-creating Information Technology Based on Laws for Software Evolvability*. Koppa, 2009.

[3] H. Mannaert, J. Verelst, and K. Ven, "The transformation of requirements into software primitives: Studying evolvability based on systems theoretic stability," *Science of Computer Programming*, vol. 76, no. 12, pp. 1210 – 1222, 2011, special Issue on Software Evolution, Adaptability and Variability. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016764231000208X

[4] G. Oorts, P. Huysmans, P. D. Bruyn, H. Mannaert, J. Verelst, and A. Oost, "Building evolvable software using normalized systems theory: A case study," *2014 47th Hawaii International Conference on System Sciences*, vol. 0, pp. 4760–4769, 2014.

[5] H. Mannaert, J. Verelst, and K. Ven, "Towards evolvable software architectures based on systems theoretic stability," *Software: Practice and Experience*, vol. 42, no. 1, pp. 89–116, 2012. [Online]. Available: http://dx.doi.org/10.1002/spe.1051

[6] P. D. Bruyn, G. Dierckx, and H. Mannaert, "Aligning the normalized systems theorems with existing heuristic software engineering knowledge," in *Proceedings of The Seventh International Conference of Software Engineering Advances (ICSEA)*, ser. ICSEA '12, Lisbon, Portugal, 2012, pp. 84–89.

[7] P. D. Bruyn, P. Huysmans, G. Oorts, D. V. Nuffel, H. Mannaert, J. Verelst, and A. Oost, "Incorporating design knowledge into software development using normalized systems," *International Journal On Advances in Software*, vol. 6, no. 1&2, pp. 181 – 195, 2013.

[8] M. Alavi and D. E. Leidner, "Review: Knowledge management and knowledge management systems: Conceptual foundations and research issues," *MIS Quarterly*, vol. 25, no. 1, pp. pp. 107–136, 2001. [Online]. Available: http://www.jstor.org/stable/3250961

 ISBN: 978-1-61208-367-4

Towards Task Allocation in Global Software Development Projects

Sajjad Mahmood, Sajid Anwer, Waleed Umar, Mahmood Niazi, Mohammad Alshayeb
Department of Information and Computer Science
King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia
Emails: {smahmood, g201303950, g201207040, mkniazi, alshayeb}@kfupm.edu.sa

Abstract— **Global Software Development (GSD) initiative aims to facilitate software development process by providing access to skilled workers at a relatively low cost and a 24/7 software development model. Previous work suggests that a significant number of companies that have tried GSD have failed to realize the anticipated benefits, which have resulted in poor outsourcing relationships, high costs and overall poor software products. Task allocation is one of the critical factor for a successful GSD project as project managers not only need to consider their workforce but also need to take into the account the characteristics of the geographically distributed sites involved in a project. In this paper, we present a task allocation process with an aim to utilize different geographically distributed sites for a GSD project. The task allocation process uses project scheduling techniques, e.g., Critical Path Method (CPM)/ Program Evaluation and Review Technique (PERT), and a multi-objective optimization technique to allocate GSD project tasks. We also present an application of the task allocation process to Obesity Health Clinic System (OHCS) case study.**

Keywords-global software development; task allocation; work distribution; software project schedules.

I. INTRODUCTION

Global Software Development (GSD) occurs where a company (client) contracts out all or part of its software development tasks to another company (vendor), who provides services for remuneration [1]. GSD has been growing steadily as a large number of organizations aim to take advantage of using highly skilled workforce at a relatively reduced cost. Furthermore, GSD has the potential to reduce project's time to market by using different time zones to organize a 24/7 development model.

A good number of organizations that have tried GSD failed to realize the expected outcomes, which resulted in misunderstanding of requirements, poor global relationships among clients and vendors, high costs and overall poor services [2][3]. These failures are usually traced back to two main causes: insufficient abilities (e.g., absence of domain knowledge, high turnover rate, etc.) at different sites, and problems at the interfaces between two distributed sites due to cultural barriers.

These issues are directly impacted by the decisions taken at the task allocation phase of a GSD project. Existing research suggests that tasks need to be allocated to different geographical sites based on a number of often conflicting objectives such as low cost [4][5], reduced development time, and increased productivity [6]; and minimum communication and coordination between different development sites [7]. For example, Tran and Latapie [8] have presented models for structuring teams and work in globally distributive projects by taking into account the dependencies between components at a higher abstraction level.

In this paper, we present a task allocation process that takes into account the interdependencies between project tasks and geographically distributed sites' capabilities. The task allocation process has two inputs, namely, GSD project tasks and number of geographically distributed sites, which have the required skills to complete the project tasks. The task allocation process uses either CPM [15] or PERT [15] to develop schedule of a GSD project. Next, the tasks are ranked based on the precedence requirements and these ranked tasks are used an input to a 'goal program', a multi-objective optimization technique, to select suitable geographical sites for a GSD project.

The rest of this paper is organized as follows: Section II reviews the related literature. In Section III, we present the task allocation model and Section IV presents a case study. We conclude the paper and discuss future work in Section V.

II. RELATED WORK

This section presents different approaches that researchers adopt for task allocation in global software development. Setamanit et al. [6] propose a simulation model based on two factors of software process model development time and productivity and it compare different task allocation strategies proposed in research against these two factors. They consider different important properties of software, i.e., coupling between activities, sites capabilities and project plan for comparing different approaches. In fact this model is proposed for comparing different site allocation techniques not allocation tasks to different sites in GSD.

 ISBN: 978-1-61208-367-4

Lamersdorf et al. [9] proposed qualitative approach intended for understanding and identifying different criteria that are practically used for task allocation in GSD. They conducted interviews from different industry peoples to collect data. They showed that type of required system and sourcing strategy mainly affects the task allocation criteria. Their study analysis shows that market proximity, turnover rate strategic planning and labor cost are main factors that need consideration during task allocation.

Lamersdorf et al. [10] presented a risk driven customizable model to suggest different task allocation approaches based on the target system and software development process model and critically evaluated this model based on risks related to tasks allocation in GSD. They also evaluated proposed approach by interviewing different peoples from Information Technology (IT) industry that are related to software development in GSD.

Narendra et al. [11] presented an integrated formal technique for analyzing all tasks and developed optimal tasks allocation model for GSD projects. The proposed model predicts estimated effort required for particular task based on the overall allocation of tasks over estimated effort and effort required to execute a particular task on particular site.

Wickramaarachchi and Lai [5] proposed a method for work distribution to different locations with an aim to minimize overhead costs. The method categorizes the offshore tasks based on software process model. It also proposes a method to distribute work to suitable tasks using work specific matrix, work dependency matrix and site dependency matrix.

Mockus and Weiss [7] proposed an approach for task allocation in GSD that mainly addresses the communication problem between sites and ultimately reduced the overall overhead in distributed development and used optimization algorithms to implement this approach. Proposed approach is well understood and easily applicable in GSD but they only consider one factor that affects GSD. As there are other factors that can affect the distributed development like cost, site capabilities and tasks dependencies and due to this limitation this approach however, cannot be used for tasks allocation in GSD.

Vathsavayi et al. [4] discuss the solution of work allocation problem in GSD using genetic algorithms. They proposed a model that take different activities of software development process as an input and find the near optimal solution. Their model is capable enough to accommodate the change management and other concerns related to project management. They considered only duration and cost factors of site for tasks allocation in GSD.

Lamersdorf and Munch [12] proposed task allocation approach that considers different factors, i.e., time zone difference, sites capabilities, labor cost and cultural issues. They developed a tool named Task Allocation based on Multiple criteria (TAMRI) using Bayesian network [12]. This model is applicable across a number of products by just modifying the underlying Bayesian approach.

Shen et al. [13] proposed approaches to solve the multi criteria task allocation problem using fuzzy numbers and linguistic scales. Linguistic scales are first used to measure quantitative properties like individual capabilities and then fuzzy numbers are used to measure and quantified these scales. Four criteria, namely, workload, familiarity with task, capability and social relationships with other team members are used to assess the individual that is suitable for this particular task.

Tran and Latapie [8] proposed a task allocation model from architectural point of view and considered the dependencies between components as criteria for task allocation. They allocated architectural integration to one site and other activities to other sites based on dependencies exist between them while other sites coordinate with the main site. The main limitation of this approach is that they consider dependencies at component level (abstract level) and dependencies inside the component can cause delay.

III. TASK ALLOCATION PROCESS

In a traditional software development environment, a project manager typically distributes work tasks among its team members who are present at a single development site. However, in GSD projects, a project manager needs to assign tasks to teams who are usually present at different geographical locations. This introduces an extra complexity at the task allocation phase of a project as one GSD vendor can be cheaper than other while another vendor might have more skilled workers. We present the task allocation decision model, as shown in Figure 1, which acts as a tool that helps managers to assign tasks in a GSD project.

The task allocation process has two inputs, namely, GSD project tasks and number of geographically distributed sites, which have the required skills to complete the project tasks. A project task is defined as a small manageable unit with a time requirement. Resource requirements for a task define the manpower required for the activity. Project tasks usually are not standalone and have precedence relationships with other tasks in a project. Furthermore, the precedence relationship also defines that what tasks can run concurrently with other tasks.

 ISBN: 978-1-61208-367-4

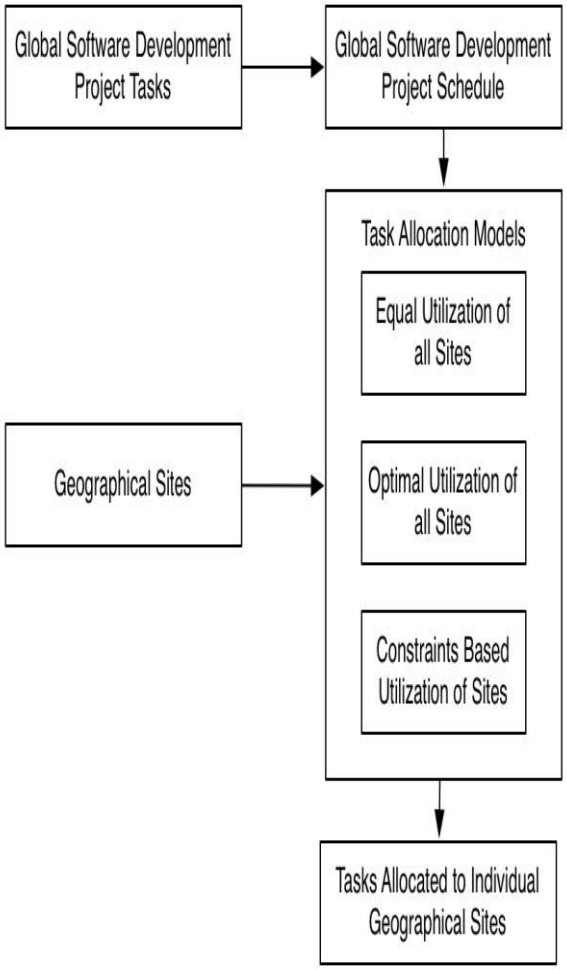


Figure 1. Task Allocation Process.

The task allocation process uses either CPM or PERT to develop schedule of a GSD project. CPM or PERT allows a project manager to define various components of a schedule such as floats, early start time, early finish time and project completion date. The task allocation process uses PERT when the end date for the activities cannot be defined but can be represented by expected probabilistic durations.

Next, the tasks are ranked based on the precedence requirements and these ranked tasks are used an input to a 'goal program' to select suitable geographical sites for a GSD project. 'Goal programming' helps achieving an optimal or near optimal solution for a set of goals.

In the following subsections, we present three models for task allocation, namely, 'equal utilization of all sites', 'optimal utilization of all sites', and 'constraints-based utilization of sites', respectively.

A. *Equal utlization of all sites*

In this model, we aim to equally distribute GSD project tasks to all geographically distributed sites. We propose to use a genetic algorithm to assign GSD project tasks, such that the standard deviation of total man-hours required at all GSD sites is minimized. The mathematical form of the proposed model as follows:

T.M.U at a site =

$$\sum_{k=0}^{n} (activity\ duration)\ x\ (Man - hours\ required\ per\ day) \quad (1)$$

where T.M.U= Total Man-hours Utilized, n is the number of the tasks to be performed at a site.

Mean =$(\sum_{l=0}^{m} \text{T. M. U at a site})$ / number of geographical sites

where m is the number of total geographical sites involved in the project.

$$\text{Variance} = \sum_{l=0}^{m}[(\text{T. M. U at a site}) - (\text{mean})]^2 \quad (2)$$

B. *Optimal utilization of all sites*

The aim of this model is to assign the tasks to the sites so that the each site is optimally utilized. The model tries to assign the various tasks to a site based on the capacity of a site and the man-hours requirement of individual tasks. The mathematical form of the proposed model as follows:

$$\text{Minimize the Goal} = (\sum_{l=0}^{m} \text{S. D. S. U}) \quad (3)$$

$$\text{S.D.S.U} = (\sum_{t=0}^{d} \text{S. D. Sites capabilities}) \quad (4)$$

where S.D.S.U= Standard Deviation of Site Utilization, S.D. = Standard Deviation, d is the duration and m are the total geographical sites.

C. *Constraints-based utilization of sites*

In this model, we enhance the 'equal utilization of all sites' and 'optimal utilization of all sites' by introducing a set of constraints. The 'utilization of sites based on constraints' allows project managers to assign tasks to a certain site. Mathematically, the objective function is defined as follows:

$$\text{Minimize} = \sqrt{\text{Variance}} + \sum_{l=0}^{m} \text{S. D. S. U} \quad (5)$$

 ISBN: 978-1-61208-367-4

IV. CASE STUDY

This section discusses an application of the proposed schedule flexibility analysis to the Obesity Health Clinic System (OHCS). The OHCS allows health team members and patients to create obesity reducing health goals. The goals are added to the 'bank of ideas' and classified under the appropriate category (for example, physical, dietary, etc.). These goals can also be customized according to individual patient needs by the health team. The OHCS also has the 'goal suggestion' feature, which helps the health team to find appropriate goals for a patient according to his health condition.

Table I presents a list of OHCS tasks, planned durations in days and their respective dependencies. Figure 2 presents the CPM network of OHCS.

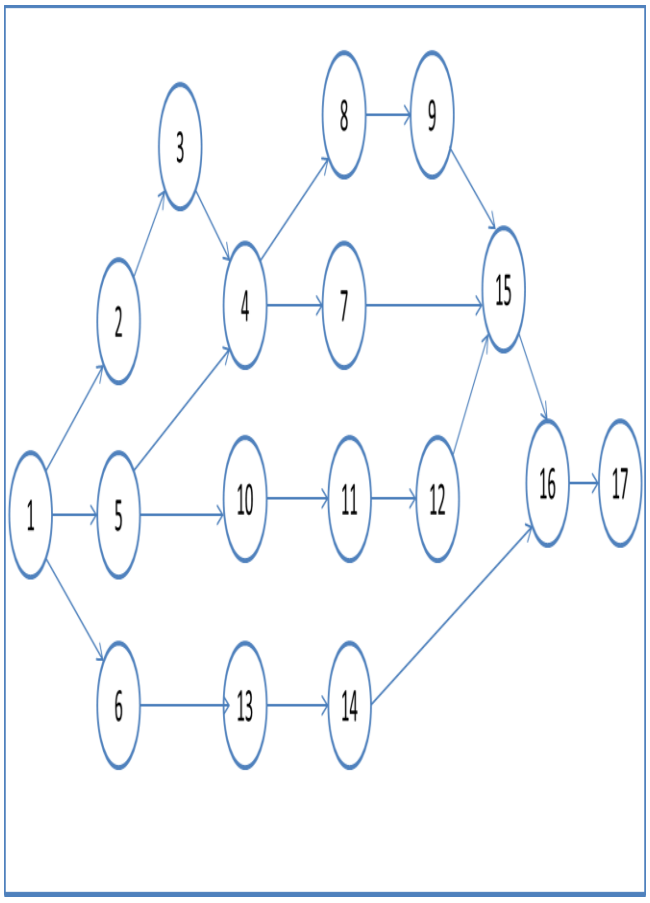


Figure 2. OHCS Activity Network.

Furthermore, Table II shows available resources (in term of man hours per day) at four geographical development sites for the OHCS project.

TABLE I. OHCS TASKS

Task ID	Activities	Duration	predecessors			Man-Hour Per Day
1	Source Node	0				0
2	Patient profile	7	1			5
3	Health Team Profile	11	2			4
4	Complete profile features	5	3	5		6
5	Database Implementation	14	1			5
6	Profile Management Screen layout	5	1			4
7	OHCS Reports	9	4			3
8	Goal Management	18	4			5
9	Goal Suggestion feature	21	8			6
10	SQL Queries	20	5			4
11	Implement Store procedures	9	10			5
12	Complete database implementation	5	11			5
13	Goal Management screen layout	5	6			4
14	Complete OHCS Screen layouts	14	13			6
15	Complete OHCS Feature and Database integration	12	7	9	12	3
16	Complete system integration	18	14	15		6
17	OHCS Deployment	6	16			6

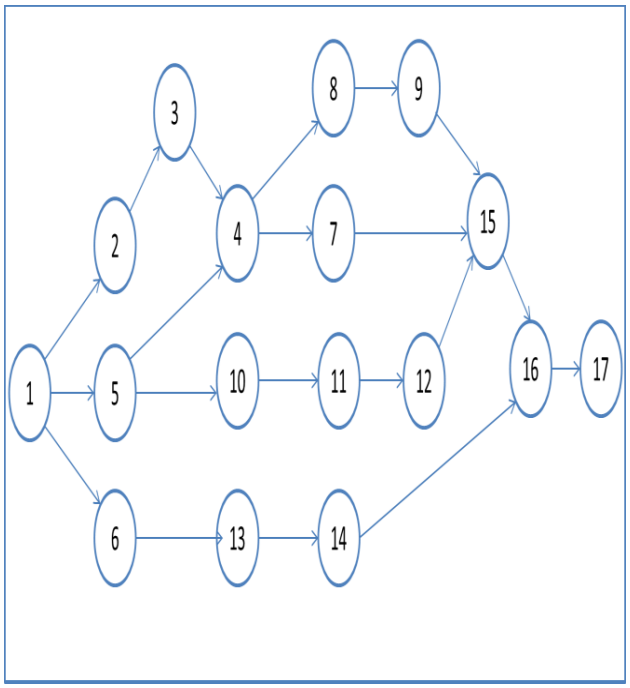


Figure 3. OHCS Activity Network.

TABLE II. OHCS GSD DEVELOPMENT SITES

GSD Sites	A	B	C	D
Resources Available (man-hour/day)	6	4	5	3

The OHCS project activates and project geographically distributed sites, presented in Tables I and II. OHCS project activities and the distributed sites are used as inputs such that all the GSD sites are optimally utilized. We used Evolver [14], a multi-objective optimization tool, to implement the three models for task allocation presented in Section III.

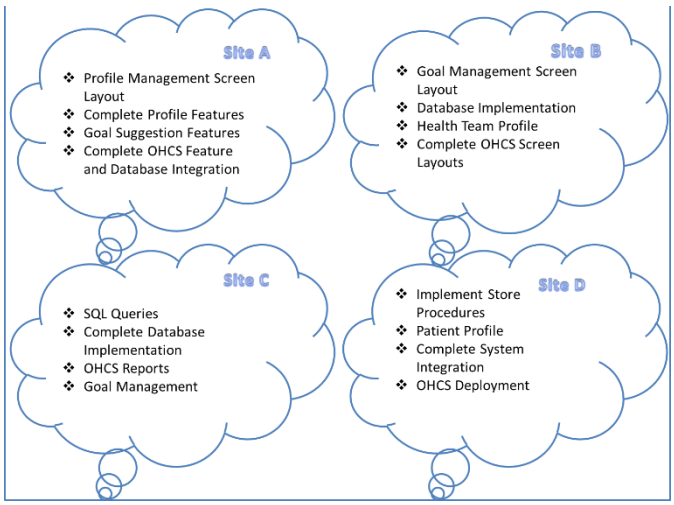


Figure 4. Equal Utilization of Four GSD Sites.

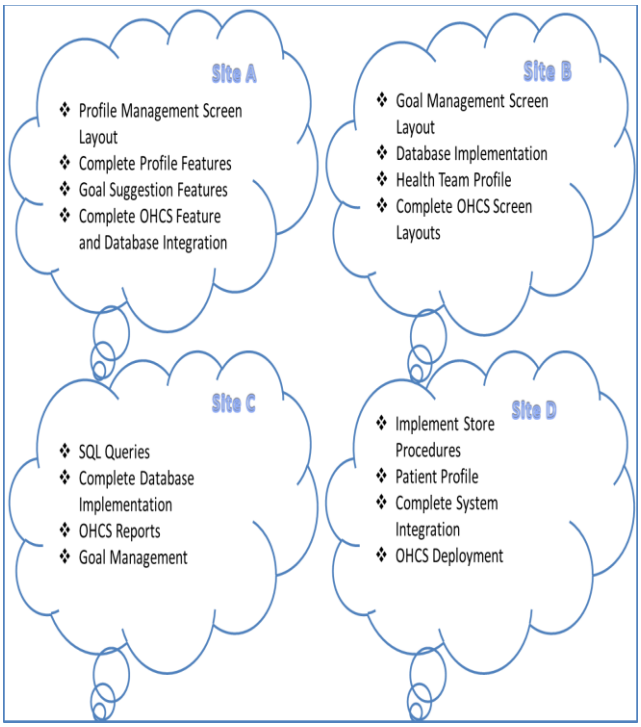


Figure 5. Optimal Utliziation of four GSD Sites.

Figure 3 shows the task allocation of OHCS project to four geographical sites using 'equal utilization of all sites' model. Figure 4 shows the task allocation of OHCS project based the 'optimal utilization of all sites' model. Similarly, Figure 5 shows the task allocation under the 'constraints based utilization of sites' model such that project managers wants to use site A's expertise in interface design and site D's expertise in database implementation.

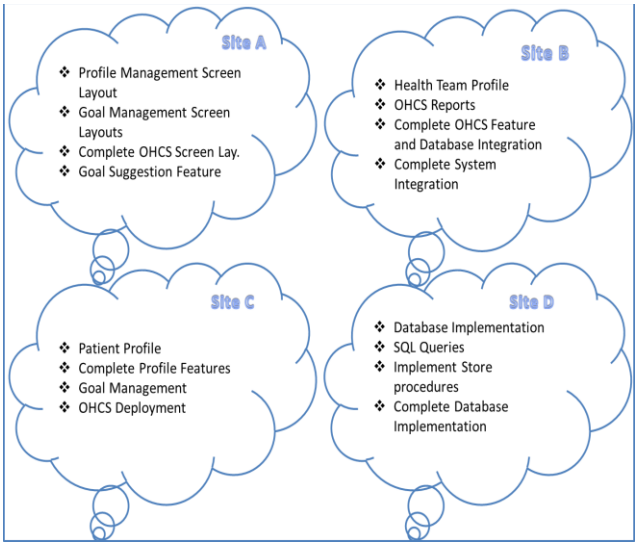


Figure 6. Utilization of four GSD Sites under Constraints.

 ISBN: 978-1-61208-367-4

V. CONCLUSIONS AND FUTURE WORK

Global software development approach is adopted by organizations with an aim to reduce development cost, improve overall software quality and increase productivity by having work carried out along the day using follow-the-sun concept. Task allocation is a key phase of GSD projects that directly impacts the benefits of adopting GSD. In this paper, we have presented a task allocation process to equally utilize different geographically distributed sites for a GSD project. The task allocation process uses a multi-objective optimization technique to allocate tasks of a GSD project.

In the future, we aim at extending the task allocation process to handle complex objective functions and improve work distribution among different sites of a GSD project. We also aim to evaluate the presented model using larger case studies.

ACKNOWLEDGMENT

The authors would like to thank King Fahd University of Petroleum and Minerals (KFUPM) for its continuous support of research. This research is supported by the Deanship of Scientific Research at KFUPM under Research Grant IN131013.

REFERENCES

[1] T. Kern, and L. Willcocks, "Exploring information technology outsourcing relationships: theory and practice", Journal of Strategic Information Systems, vol. 9, pp. 321-350, 2000.

[2] D. D. Bradstreet, and Bradstreet's Barometer of Global Outsourcing. 2000 [cited 2007 September]; Available from: <http://findarticles.com/p/articles/mi_m0EIN/is_2000_Feb_24/ai_59591405>.

[3] S. Islam, M. M. A. Joarder, and S. H. Houmb, "Goal and risk factors in offshore outsourced software development from vendor's viewpoint", In Proceedings of IEEE International Conference on Global Software Engineering (ICGSE 09), pp. 347-352, 2009.

[4] S. Vathsavayi, O. Sievi-Korte, K. Koskimies and K. Systa, "Planning global software development projects using genetic algorithms", In Search Based Software Engineering, G. Ruhe and Y. Zhang, Editors. pp. 269-274, 2013.

[5] D. Wickramaarachchi, and R. Lai, "A method for work distribution in Global Software Development", In Proceedings of IEEE 3rd International Advance Computing Conference, pp.1443-1448, 2013.

[6] S. Setamanit, W. Wakeland, and D. Raffo, "Planning and improving global software development process using simulation", In Proceedings of the 2006 international workshop on global software development for the practitioner, pp.8-14, 2006.

[7] A. Mockus, and D. M. Weiss, "Globalization by chunking: a quantitative approach", IEEE Software, vol. 18, pp. 30-37, 2001.

[8] V. N. Tran, and H. M. Latapie, "Models for structuring teams and work in globally collaborative projects", In Proceedings of IEEE International Engineering Management Conference, pp. 425-431, 2006.

[9] A. Lamersdorf, J. Munch, and D. Rombach, "A survey on the state of the practice in distributed software development: criteria for task allocation", In Proceedings of 4th IEEE International Conference on Global Software Engineering, pp. 41-50, 2009.

[10] A. Lamersdorf, J. Munch, A. F. V. Torre, and C. R. R. Sanchez, "A risk-driven model for work allocation in global software development projects", In Proceedings of 6th IEEE International Conference on Global Software Engineering, pp. 15-24, 2011.

[11] N. C. Narendra, K. Ponnalagu, Z. Nianjun and W. M. Gifford. "Towards a Formal Model for Optimal Task-Site Allocation and Effort Estimation", In Proceedings of Global Software Development Conference, pp. 470-477, 2012.

[12] A. Lamersdorf, and J. Munch, "TAMRI: A Tool for Supporting Task Distribution in Global Software Development Projects", In Proceedings of 4th IEEE International Conference on Global Software Engineering, pp. 322-327, 2009.

[13] M. Shen, G. Tzeng, and D. Liu., "Multi-criteria task assignment in workflow management systems", In Proceedings of the 36th Annual Hawaii International Conference on System Sciences, pp. 9-17, 2003.

[14] www.palisade.com/evolver., last accessed June, 2014.

[15] K. Schwalbe, "Information technology project management", Course Technology, 6th Edition, 2010.

 ISBN: 978-1-61208-367-4

Combining MARTE-UML, SysML and CVL to Build Unmanned Aerial Vehicles

Paulo Gabriel Gadelha Queiroz
Departamento de Ciências Exatas e Naturais - DCEN
Universidade Federal Rural do Semi-Árido
Mossoró, Brazil
Email: pgabriel@ufersa.edu.br

Rosana Teresinha Vaccare Braga
Instituto de Ciências Matemáticas e de Computação - ICMC
Universidade de São Paulo
São Carlos, Brazil
Email: rtvb@icmc.usp.br

***Abstract*—Several methodologies have been proposed in the last decades to improve the quality of critical embedded systems and, at the same time, keep costs and schedule compatible with project plans. In particular for Unmanned Aerial Vehicles (UAV), approaches such as Product Line Engineering (PLE) and Model-Driven Engineering (MDE) offer an interesting solution to reduce development complexity and are being widely used in various academic research and industrial projects. This paper presents an approach combining PLE and MDE to develop families of Unmanned Aerial Vehicles. In this approach, we propose the use of SysML and MARTE UML profile to support requirements specification, design, validation, simulation and eventual code generation. Additionally, we propose the use of the Common Variability Language (CVL) to support the transformations of the generic product line models into specific product models, aiming at achieving a high degree of reuse. Additionally, this paper proposes a process to use the above mentioned modeling techniques to produce family models and a method to use these artifacts to generate product members. Finally, we illustrate the various concepts presented in the proposed methodology by means of a UAV case study.**

Keywords–Product Line; Model-Driven Development; Safety-Critical Systems.

I. INTRODUCTION

Embedded Systems are components integrating software and hardware jointly and specifically designed to provide given functionalities [1]. Safety-Critical Embedded Systems (SCES), in particular, are embedded systems whose failure could result in loss of lives or on significant environmental or property damage. SCES are common in medical devices applications, aircraft flight control systems, weapons, and nuclear systems. Aircraft flight control systems, for example, must present failure rates as low as a serious fault per 10^8 flight hours [2] and other complex constraints and requirements like cost-effectiveness, time to market, fast evolving environment, reliability, security, availability, criticality, reactivity, autonomy, robustness, and scalability [3], which impose overhead costs on the development. In the SCES domain, we focus on Unmanned aerial vehicles (UAV), which can be defined as airplanes that fly without the need of a human pilot, accomplishing a pre-established mission.

The coming generations of SCES, like UAVs, must meet the new expectations created by hardware evolution like the increase in computational power of processors and the corresponding decrease in size and cost that lead to the increase of users expectations for new functionalities and have allowed moving more and more functionality to software [4]. Therefore, to overcome these challenges and to fulfill the requirements and constraints mentioned above, we need new efficient and flexible development methodologies and tools that can reduce UAV production complexity.

The use of Product Line Engineering (PLE) [5] has proven to be a good alternative to reduce system costs and time to market, as well as to increase system reliability through the assembling of reusable and extensively tested resources. Model-Driven Engineering (MDE) [6] is also used in this context, producing models in higher abstraction levels and allowing automatic generation of products through model transformations.

The motivations for this work have arisen after the creation of a Product Line (PL) workgroup in the National Institute of Science and Technology - Critical Embedded Systems (INCT-SEC) [7] project, whose goal was to create methodologies and tools to develop, among others, families of UAVs. The first family, called Tiriba, was developed by the AGX Company [8] in partnership with INCT-SEC. During our participation in this workgroup we performed a systematic review of the literature to find gaps in existing methodologies and approaches that combine MDE and PLE to develop SCES, as well as a study using three other UAV existing examples [9][10][11] to build a family of UAVs and validate our approach. These studies culminated with the approach proposed in this paper, whose main goal is to build a family of UAV using a combination of both PLE and MDE techniques. The novelties of the proposed approach are the use of MDE in both Domain and Application Engineering and the management of both software and hardware variabilities, accompanied by Verification and Validation (V&V) activities during the whole cycle. For an effective use of MDE, we propose the use of a subset of the Systems Modeling Language (SysML) [12] and the UML profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) [13] to enable model transformations in the Domain Engineering (DE) phase. We also propose the use of the Common Variability Language (CVL) [14] to manage system variabilities and enable model transformations during the Application Engineering (AP) phase. Finally, we illustrate the various concepts present in the proposed approach by means of a UAV family case study. It is worth to mention that this approach is an extension of the work presented in [15], with the addition of CVL to manage variabilities, the safety analysis activity and the evolution of the case study.

The rest of this paper is organized as follows: Section II presents a background summary of Product Line Engineering

 ISBN: 978-1-61208-367-4

and variability management; Section III summarizes related works; Section IV presents the proposed approach; Section V illustrates the proposed approach by means of a UAV product line case study; lastly, Section VI presents the conclusions of this paper.

II. BACKGROUND

Product Line Engineering is an approach that enables organizations to develop, deliver and evolve an entire Product Line portfolio, through each stage of the development life cycle, with much higher degrees of efficiency compared to developing single systems [16]. The products of a PL differ from each other in terms of features, which are user-visible aspects or characteristics of a software system or systems [17]. As expected, the costs, in terms of time and money spent, to build a PL is higher in comparison with the costs to build a single system, because among other things PLE is done in two stages: Domain Engineering (DE), which is the development of a series of generic artifacts to the PL; and Application Engineering (AE), in which the application engineer uses the artifacts developed in DE to assemble products of the line, known as members.

During DE, a general architecture for the PL is defined, from which various products can be generated. Despite the higher cost, Weiss and Lai [18] claim that the construction of a PL is justified if at least three systems generated from the PL are derived. Since UAV are often manufactured, distributed in large scale and present significant variability in terms of hardware and applications, it can be expected that the use of PLE is advantageous to them.

On the other hand, in Model-Driven Engineering, the software complexity concentrates on high level models and not in the code, which can be automatically generated from the models. Furthermore, system quality can be improved with the use of V&V methods [19]. According to model-based approaches, models become part of the final product and most of the development complexity shall belong to the transformations that should be used to automatically or semi-automatically produce code. To successfully use MDE techniques to model a UAV PL, we propose the use of SysML, MARTE and CVL.

SysML [12] is a general-purpose modeling language for systems engineering applications, which reuses a subset of the Unified Modeling Language (UML) [20] and provides additional extensions. SysML supports the specification, analysis, design, verification, and validation of a broad range of complex systems and is used to model a wide range of industrial and academic systems [21]. As SysML is an UML extension, there is a compatibility of tools and concepts, which can reduce the learning time. We also want to propose an approach that can be adopted using free tools, which are abundant for UML-based languages.

MARTE [13] is an UML profile that provides capabilities for model-driven development of Real Time and Embedded Systems (RTES). It provides support for specification, design, and verification/validation stages [13]. MARTE is also used to model a wide range of industrial and academic systems [22]. We adopt it in our approach because UAVs have many real time constraints that need to be checked by model simulation in early development stages, to improve product quality.

Even though some authors consider MARTE and SysML profiles incompatible, by using the MADES methodology [3] recommendation we can avoid conflicts related to the two profiles by not mixing SysML and MARTE concepts in the same diagram, but instead focusing on a refinement scheme. Therefore, as presented later, SysML is used for initial requirements and functional description, while MARTE is utilized for the enriched modeling of the global functionality and execution platform/software modeling.

Another resource that can be useful to improve the application of MDE techniques to build the UAV PL is CVL [14], which is a separate and generic language to define variabilities. CVL semantics are defined as a transformation of an original model (e.g., a product line model) into a configured, new product model. CVL combines user-centric feature diagrams with an automation-centric approach to the production of product models. In CVL, the focus is on specifying variability in a model separate from the base product line models. A base model is an instance of any metamodel conforming to Meta Object Facility (MOF) [23]. The base models are produced in domain engineering in our case. There may be several variability models applied to the same base product line model and the base model is unaware of the variability models (there are only links from the CVL model to the base model). Several product resolutions can apply to the same variability model. CVL is executable, i.e., after specifying the resolution of variabilities, a CVL tool can automatically derive the specific product model.

The core concept of CVL is substitution. Models are assumed to consist of model elements in terms of objects that are related by means of references. The CVL model points out model elements of the base PL model and defines how these model elements shall be manipulated to yield a new product model. There are three kinds of substitution: value substitution, reference substitution and fragment substitution. A substitution replaces base model elements named as placement by base model elements named as replacement [14]. CVL can represent variabilities through the concepts of Variation point, Substitution, Existence, Value assignment, Variability specification, and Choice, among others.

III. RELATED WORKS

While there is a large number of researches who make use of either PLE or MDE for safety-critical embedded systems, due to space limitations, it is not possible here to give an exhaustive description, so we only provide a brief summary of works that combine PLE and MDE to build safety-critical embedded systems, similarly to the approach proposed in this work.

The work presented by Polzer et al. [24] is concerned with variability in control systems software, where a model-based PL engineering process using Rapid Control Prototyping system is combined with MDE techniques. The authors modularize the components parameterization in a separate setup, which is isolated from the model that defines the behavior of the controller. Simulink [25] and Pure::variants [26] are used for modeling and automatic code generation. It is observed

 ISBN: 978-1-61208-367-4

that this work is done with proprietary tools and modeling techniques like Matlab building blocks that although efficient for the project description, are not ideal for requirements modeling and communication with the final user, which goes against the purpose of this paper.

Regarding the development of UAV product lines, there are approaches such as Product Line on Critical Embedded Systems (ProLiCES) [27] and SysML-based Product Line Approach for Embedded Systems (SyMPLES) [28]. Even though they were not defined for UAV, the authors used a UAV case study to illustrate their approaches. ProLiCES creates a parallel path in the process to handle the PL domain engineering and also proposes the use of Matlab/Simulink as a Model-Driven Development (MDD) technique, which limits requirements analysis and concentrates the MDD only in one step of the process. SyMPLES is an approach for PL application in embedded systems through the extension of SysML language to include variability together with a development process, but in this study the authors do not distinguish between the characteristics of hardware and software and focus on the use of SysML for the architecture description.

Svendsen et al. [29] present a case study for creating a PL for the train signaling domain. The Train Control Language (TCL) is a Domain-Specific Language that automates the production of source code for computer-controlled train stations, also using CVL. However, their approach presents just the variability management through CVL, which consists of a portion of the system product line development process.

In the work presented by Haber et al. [30] the authors focus on variability management in all development phases using Matlab/Simulink. They propose a modular variability modeling approach based on the concept of delta modeling. A functional variant is described by a delta encapsulating a set of modifications. A sequence of deltas can be applied to a core product to derive the desired variant. The authors illustrate the approach by presenting a prototypical implementation.

Finally, we highlight the Cardiac Pacemaker PL described by Huhn and Bessling [31], where they present how to specify the PL and its products by means of CVL. CVL enforces a strict structuring of the product models (done in SCADE) that reflects the substitution concepts used to describe variability.

The approach we propose in this paper is different from the above mentioned related works, as it focuses on the PL definition, modeling both hardware and software variabilities. We propose the use of MDE, like automatic generation of hardware descriptions and embedded software from high level models, for rapid design and specification of SCES. Furthermore, we propose the use of free tools for MDE, the use of UML as it is an extensively used modeling language and the use of CVL to model variability.

IV. PL APPROACH

Figure 1 illustrates our proposed Product Line approach. Notice that the approach addresses both hardware and software variability with its underlying requirement dependencies. To reduce both domain and application engineering complexity, we propose the use of UML based models, in particular SysML and MARTE. Despite being the most widely used modeling language, UML is generally easier to understand than Matlab blocks.

The strengths of the proposed methodology are the use of MDE in both Domain and Application Engineering phases, with focus on model-to-model transformations in requirements, analysis and design activities, especially for the purpose of modelling and generating application variants. This is different from most of the PL methodologies, which focus on model-to-text transformations just in the Application Engineering phase, through the use of application generators. The use of MDE has also the advantage to promote the possibility to use Model-Based Test [32] in early design stages, which can substantially reduce the V&V costs and effort [33]. Safety analysis [34] techniques is also recommended in early design stages, especially for certification purposes, but this is out of the scope of this paper.

As seen in Figure 1, the approach is divided into two interdependent phases, Domain Engineering and Application Engineering, which are common in consolidated methods like the Framework for Software Product Line Practice [16] and the PLUS method [35]. During the DE phase, the high level system models are carried out using SysML and CVL, which are exemplified later in Section V. After the system PL specification (user requirements, specification and related hardware/software variability specification), underlying model transformations (model-to-model and model-to-text transformations) are used to produce models for the subsequent design phases including MARTE profile. The next design phases include verification, hardware descriptions of modeled targeted architecture and generation of platform specific embedded software from platform independent software specifications. For implementing model transformations in the case study, we use the Eclipse Modeling Platform (EMF) [36]; the Papyrus [37] modeling tool, which is a UML modeler that enables model transformations, code generation and validation; and the CVL Eclipse Plug-in [38] as the engine for the transformations of product line models into specific product models. The proposed approach is not limited to these tools, therefore the choice of the modeling tool is up to the user, the only requirement is to support MARTE and SysML metamodels.

Another important factor to be noted is that in the UAV domain, hardware variability could impact directly on software requirements and vice versa. For example, consider the following system requirement: *the system should allow the user to choose between broadcasting the images to the ground control station in real time or to recording a video (in flash memory, for example)*. In that case, the UAV hardware must include a camera. Moreover, for each new sensor added, their corresponding software drivers must also be added. Another highlight is the continuous feedback in the artifacts repository, in which we can store any kind of artifact from both hardware or software types. As a repository to store hardware artifacts, we refer in a logical level, to a hardware models repository (the same repository to store software artifacts). This feedback can come from updates in DE or from new different requirements elucidated from new members modeled in the AE. The feedback is represented by both dashed arrows and double-headed arrows.

 ISBN: 978-1-61208-367-4

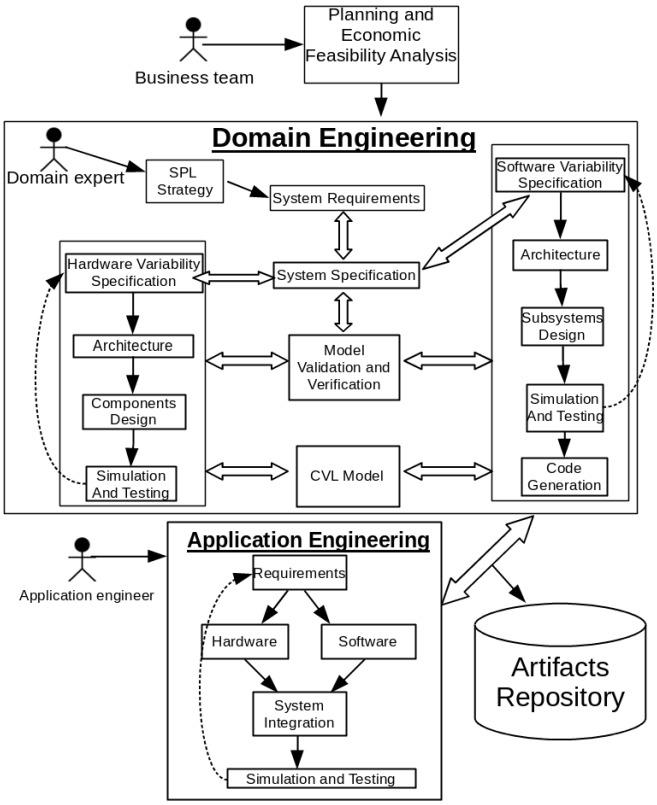


Figure 1: Overview of the proposed approach.

A. Domain Engineering

Before the Domain Engineering takes place, a business team performs an economic feasibility analysis of the PL, which will indicate whether or not it is worth to be developed. If the PL is feasible, then we start the Domain Engineering by modeling requirements in the system abstraction level, as detailed below. It is out of the scope of this work to propose domain analysis techniques, as they can be easily found in the literature. So, existing techniques such as those mentioned in the survey by Prieto-Diaz and Arango [39] can be used.

The Domain Engineering is performed by the domain expert, who should first define the PL strategy, i.e., he must decide whether to use a proactive or reactive approach [40]. Regardless of the strategy chosen, to model variabilities we propose the use of CVL. Since CVL replaces values and sets of model elements, by executing CVL we can add, remove or replace functionality. To use CVL, according to Svendsen et al. [29] the domain expert has three options for choosing a base model: the first is a model with maximum set of features included, meaning a complete model where CVL can remove features to get a specific product model; the second is a model with a minimum set of features included in the model itself, and other fragments in other library models, then the product models will be generated by adding features to the base model; and the third is to choose a base model that has neither maximum nor minimum, but somewhere in between, so this base model can be, for instance, the base model that is most similar to the majority of the product models, or a base model that is tailored for teaching purposes.

CVL proposes a model with two parts: the Feature Specification Layer (FSL) and the Product Realization Layer (PRL). The FSL resembles a feature diagram [17], while the PRL connects the FSL to the base model, for example by substitutions. We also suggest that domain experts develop the FSL incrementally: first, create a high level system FSL, then add the software and the hardware features. Hardware variability management should concern the impact evaluation of hardware variabilities on software requirements, in the same way that software variability management should concern the impact evaluation of software variabilities in hardware requirements. To manage this impact, the dependencies between hardware and software features should be defined.

As we propose the use of CVL in conjunction with a subset of MARTE and SysML models to describe the PL, the domain expert should prepare a PRL that corresponds to each SysML and MARTE models, so that after the variability resolution, all models created for the PL are automatically adapted for each product.

To complement the system requirements definition, a SysML requirements diagram should be created, where distinguishing between functional and non-functional requirements is recommended. As illustrated in Figure 1, the artifact repository is updated during the PL life cycle for both domain experts and application engineers. So, if concrete products have new requirements not covered by the PL, the requirements diagram can be further reviewed to include them.

Following the system requirements activity, we proceed to the system specification activity, where we produce use case scenarios and a system domain model. The use case is described using traditional UML Use Case Diagrams. The system domain model can be modeled using a class diagram, in which the concepts are represented by pseudo-classes. These two modeling concepts are strongly related to the functional high level specification described subsequently. While the use case is needed to obtain a SysML block diagram, as explained below, the domain model is used to communicate with domain experts for a better understanding about the domain, for validating the specification and for a future definition of a domain specific language by means of a UML profile, for example.

To finalize this system initial description, each use case is converted into a SysML block (or internal block), for example by applying the MADES methodology [3], with the difference that a mandatory use case is converted to a mandatory block, an optional use case is converted to an optional block, and an alternative use case is converted to an alternative block. After including all the developed artifacts in the repository, we can continue the PL development by going to hardware or software abstract levels or even to both in parallel.

Following this initial system specification, the development can evolve into two parallel paths, as illustrated in Figure 1. The first path starts at hardware specification, architecture definition, design of the components and simulation. The second path goes through software variability specification and management, architecture definition, subsystems design, simulation, testing, and code generation.

The designer can move to the partitioning of the system in question: depending upon the requirements and resources in

 ISBN: 978-1-61208-367-4

hand, he or she can determine which part of the system needs to be implemented in hardware or software. It is possible, although it could substantively increase SPL costs, to improve safety by implementing system features in a redundant way, i.e., whenever possible, to provide features implementations in both hardware and software. Thus, it becomes part of the Application Engineering to decide if the features implementation component should be integrated in the product by software or hardware.

Since the proposed approach focuses on UAV, V&V activities should be executed in each stage. It is important to notice that on hardware and software paths a more detailed specification takes place by the eventual allocation with schedulability and underlying model transformations (model-to-model and model-to-text transformations) that are used to bridge the gap between these abstract design models and subsequent design phases. These phases include verification, hardware descriptions of modelled target architecture and generation of platform specific embedded software from independent architectural software specifications.

For a description of the different steps related to each design level by means of MARTE concepts, see the work of Quadri et al. [3], which can be adapted to this approach by creating a CVL model for the base models representing the hardware and software specification, architectural definition, components and subsystems design and simulation. It is also important to perform validation activities in every model to ensure they correspond to the requirements and are traceable to each other. At the end of this phase, our repository contains all the artifacts and the domain engineering is ended.

B. Application Engineering

Application Engineering corresponds to configuring a product by assembling reusable artifacts from the repository. This step is the responsibility of the application engineer, who elicits the particular system requirements. By using our approach, the application engineering phase is simplified and reduced to the definition of the resolution model, which consists of selecting the desired features for the PL member. So, the application engineer can choose which substitutions to execute, and then execute the CVL model that will generate specific products (i.e., specific models). To conclude this step, it is necessary to conduct simulation and testing also on the target system to validate it.

V. Case Study

To illustrate the use of the proposed methodology, we present the initial development of a UAV PL. Through this example, we aim to show the power of CVL to manage hardware and software variabilities. We assume that the PL economic feasibility analysis indicated it is worth to be developed. The domain expert defined the following strategies: reactive approach with the base model with maximum set of features. We have chosen Tiriba as a starting point, but intend to include other UAVs in future works.

In Figure 2, we illustrate part of the SysML requirements diagram with the maximum set of features. The next step is to create the CVL model, which comprises FSL and PRL, for this base model. For the management of both hardware and

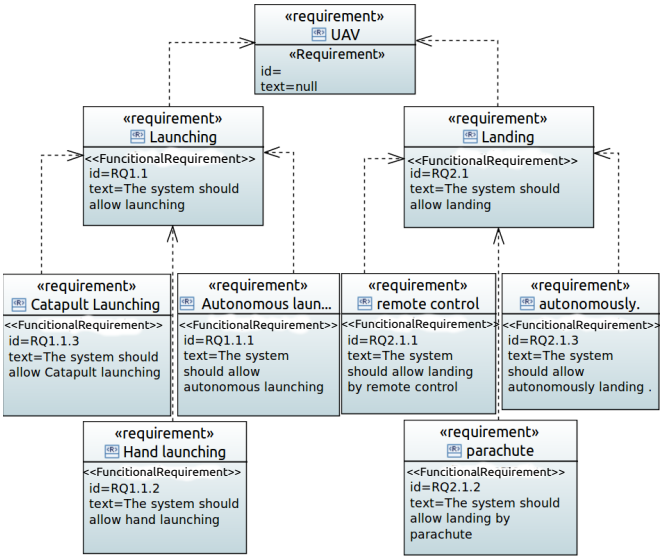


Figure 2: Part of the SysML requirements diagram for the UAV Product Line.

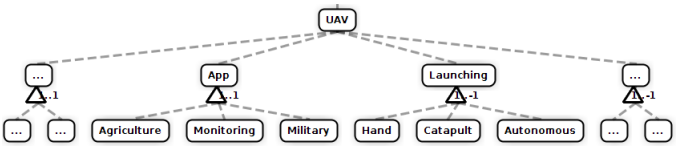


Figure 3: Part of the system high level FSL for the UAV Product Line.

software features, we should create a hardware and software FSL, which are modelled from the system high level feature diagram, as illustrated in Figure 3. To finish this first part, we define the PRL, which connects the feature specification layer to the base model and the substitutions, like illustrated in Figure 4 through an architectural view. Observe that the base model has the maximum set of features, thus a subtractive strategy is used for most parts, and sometimes a substitution.

Supposing that the same process is done for the other models proposed and all the models are stored in a repository, all the application engineer needs to do is to create the resolution model, by selecting the required features for the product. A possible resolution model with the positive choices for *Autonomous Navigation*, *Manual Control*, *Agrochemical Sprayer* and *Video Camera* is illustrated in Figure 5. After executing the CVL engine, the PRL is transformed into the resulting product model presented in Figure 6.

VI. Conclusions and Future Works

This paper aimed to present an approach for UAV product lines modeling with the use of MDE techniques in both Domain and Application Engineering, as well as software and hardware variability management. To fulfill this objective we have used a subset of UML profiles like SysML, MARTE in combination with the CVL. The use of the proposed approach in the UAV domain can bring the benefits of PL and MDE techniques, such as reducing system costs and time-to-market

 ISBN: 978-1-61208-367-4

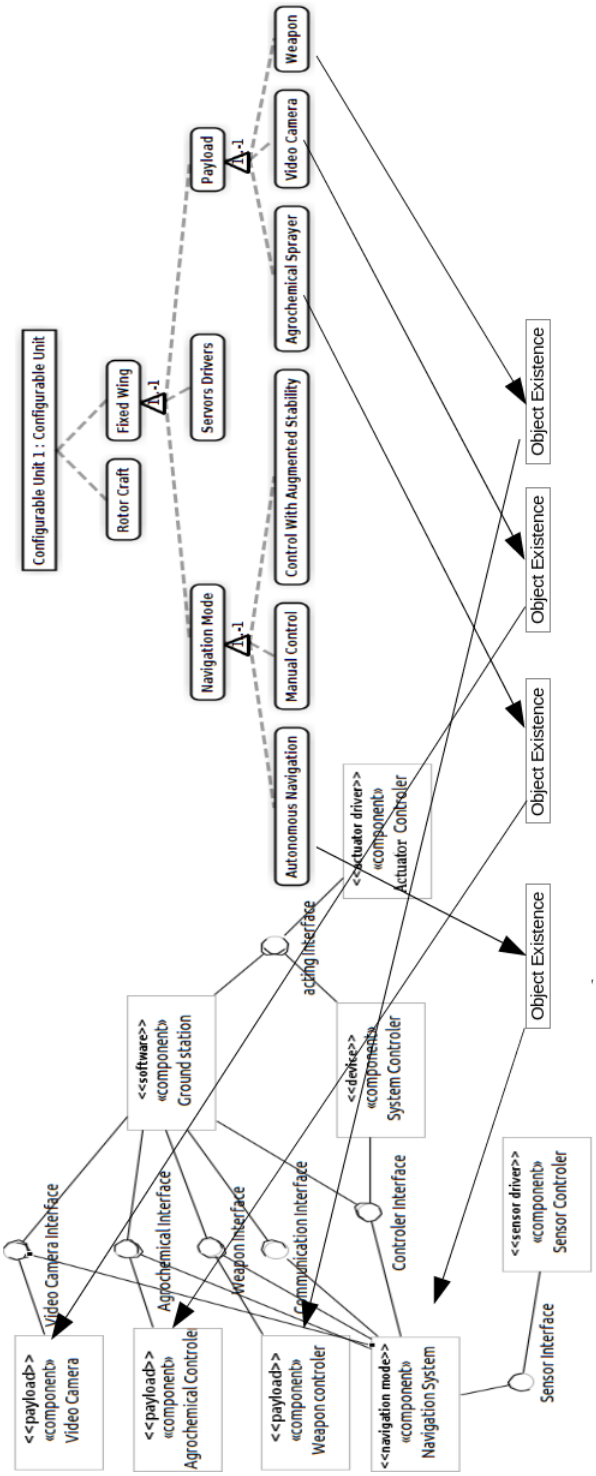


Figure 4: Part of the PRL for the UAV Product Line.

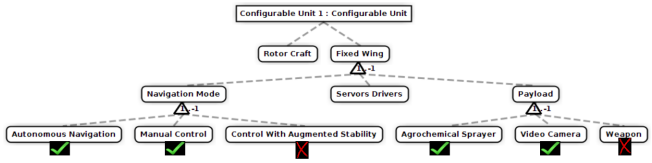


Figure 5: Part of the UAV resolution model.

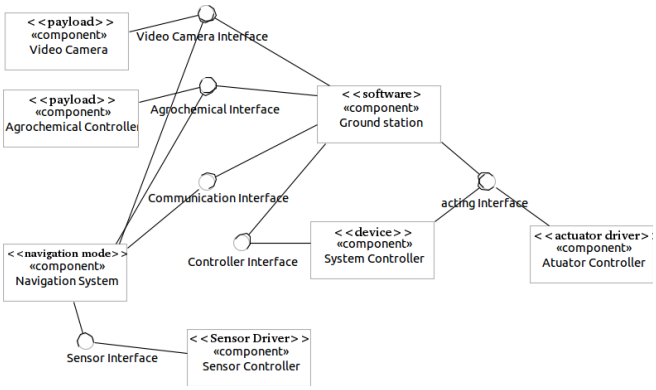


Figure 6: Part of the UAV product model after processing the variabilities.

and increasing system reliability. Finally, the most important steps, models and concepts from the proposed approach have been illustrated by a UAV product line case study. The main limitation of our approach is the lack of definition of a UAV UML profile to improve model-to-code transformation. Therefore, for future work we propose to define a UAV UML profile and to evaluate the proposed approach by means of an experiment to compare our approach with others in terms of efficiency and usability.

Acknowledgments

The authors would like to thank CAPES (*Coordenação de Aperfeiçoamento de Pessoal de Nível Superior*) for financial support received during the development of this work.

REFERENCES

[1] J. Sifakis, “Embedded systems - challenges and work directions,” in Principles of Distributed Systems, 8th International Conference, Grenoble, France, ser. Lecture Notes in Computer Science, T. Higashino, Ed., vol. 3544. Springer, Dec. 2004, pp. 184–185.

[2] I. Moir, Civil Avionics Systems, ser. Aerospace Series (PEP). John Wiley Sons Ltd, 2006.

[3] I. R. Quadri, A. Sadovykh, and L. S. Indrusiak, “MADES: a SysML/MARTE high level methodology for real-time and embedded systems,” in ERTS2 2012: Embedded Real Time Software and Systems, Feb. 2012, pp. 1–10.

[4] J. R. Burch, R. Passerone, and A. L. Sangiovanni-Vincentelli, “Using multiple levels of abstractions in embedded software design,” in Embedded Software, ser. Lecture Notes in Computer Science, T. A. Henzinger and C. M. Kirsch, Eds. Springer Berlin Heidelberg, Sep. 2001, vol. 2211, pp. 324–343. [Online]. Available: http://dx.doi.org/10.1007/3-540-45449-7_23 [retrieved: August, 2014]

[5] K. Pohl, G. Böckle, and F. J. v. d. Linden, Software Product Line Engineering: Foundations, Principles and Techniques. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

 ISBN: 978-1-61208-367-4

[6] S. Kent, "Model driven engineering," in Integrated Formal Methods, ser. Lecture Notes in Computer Science, M. Butler, L. Petre, and K. Sere, Eds. Springer Berlin Heidelberg, Apr. 2002, vol. 2335, pp. 286–298. [Online]. Available: http://dx.doi.org/10.1007/3-540-47884-1_16 [retrieved: August, 2014]

[7] INCT-SEC, "Sistemas Embarcados Críticos: aplicações em segurança e agricultura," 2008. [Online]. Available: http://www.inct-sec.org [retrieved: August, 2014]

[8] AGX Tecnologia Ltda, 2014. [Online]. Available: www.agx.com.br [retrieved: August, 2014]

[9] GISA-Grupo de Interesse em Sisvants e Aplicaes, 2014. [Online]. Available: http://gisa.icmc.usp.br/site/ [retrieved: August, 2014]

[10] SLUGS-Santa Cruz Low-cost UAV GNC System , 2014. [Online]. Available: http://slugsuav.soe.ucsc.edu/ [retrieved: August, 2014]

[11] Ardupilot , 2014. [Online]. Available: http://ardupilot.com/ [retrieved: August, 2014]

[12] Object Management Group, OMG Systems Modeling Language (OMG SysML), V1.3, 2012.

[13] Object Management Group*, UML Profile for MARTE (Modeling and Analysis of Real-Time and Embedded Systems) 1.1, 2011, OMG doc. http://www.omg.org/spec/MARTE/1.1/ [retrieved: August, 2014].

[14] O. Haugen, A. Wasowski, and K. Czarnecki, "Cvl: Common variability language," in Proceedings of the 16th International Software Product Line Conference - Volume 2, ser. SPLC '12. New York, NY, USA: ACM, 2012, pp. 266–267. [Online]. Available: http://doi.acm.org/10.1145/2364412.2364462

[15] P. G. G. Queiroz and R. T. V. Braga, "A critical embedded system product line model-based approach," in SEKE-26: Proceedings of the The 26th International Conference on Software Engineering and Knowledge Engineering. London, UK: Springer, Jul. 2014, pp. 71–75.

[16] Northrop, L. M. et al., "A Framework for Software Product Line Practice, Version 5.0," 2009. [Online]. Available: http://www.sei.cmu.edu/productlines/framework.html [retrieved: August, 2014]

[17] K. C. Kang, J. Lee, and P. Donohoe, "Feature-Oriented Product Line Engineering," IEEE Software, vol. 19, no. 4, Aug. 2002, pp. 58–65.

[18] D. M. Weiss and C. T. R. Lai, Software Product-line Engineering: A Family-based Software Development Process. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[19] L. Belategi, G. Sagardui, and L. Etxeberria, "Model based analysis process for embedded software product lines," in Proceedings of the 2011 International Conference on Software and Systems Process, ser. ICSSP '11. New York, NY, USA: ACM, May. 2011, pp. 53–62. [Online]. Available: http://doi.acm.org/10.1145/1987875.1987886 [retrieved: August, 2014]

[20] O. M. Group, "OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2," Tech. Rep., nov 2007. [Online]. Available: http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF [retrieved: August, 2014]

[21] P. Baker, S. Loh, and F. Weil, "Model-Driven Engineering in a Large Industrial Context - Motorola Case Study," in Proceedings of the 8th International Conference on Model Driven Engineering Languages and Systems, ser. MoDELS'05. Berlin, Heidelberg: Springer-Verlag, Oct. 2005, pp. 476–491.

[22] M. Z. Iqbal, A. Arcuri, and L. Briand, "Environment modeling with uml/marte to support black-box system testing for real-time embedded systems: Methodology and industrial case studies," in Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems: Part I, ser. MODELS'10. Berlin, Heidelberg: Springer-Verlag, Oct. 2010, pp. 286–300. [Online]. Available: http://dl.acm.org/citation.cfm?id=1926458.1926486 [retrieved: August, 2014]

[23] omg, Meta Object Facility (MOF) Core Specification Version 2.0, 2006. [Online]. Available: http://www.omg.org/cgi-bin/doc?formal/2006-01-01

[24] A. Polzer, S. Kowalewski, and G. Botterweck, "Applying software product line techniques in model-based embedded systems engineering," in Model-based Methodologies for Pervasive and Embedded Software (MOMPES 2009), Workshop at the 31st International Conference on Software Engineering (ICSE 2009), vol. 0. IEEE Computer Society, May May. 2009, pp. 2–10.

[25] O. Beucher, MATLAB und Simulink (Scientific Computing). Pearson Studium, 08 2006.

[26] Pure Systems, "pure::variants," 2012.

[27] Braga, R. T. V. et al., "The prolices approach to develop product lines for safety-critical embedded systems and its application to the unmanned aerial vehicles domain." CLEI Electron. J., vol. 15, no. 2, May 2012, pp. 1–13. [Online]. Available: http://dblp.uni-trier.de/db/journals/cleiej/cleiej15.html#BragaBJMNB12 [retrieved: August, 2014]

[28] R. F. Silva, V. H. Fragal, E. A. de Oliveira Junior, I. M. de Souza Gimenes, and F. Oquendo, "SyMPLES - A SysML-based Approach for Developing Embedded Systems Software Product Lines." in ICEIS (2), S. Hammoudi, L. A. Maciaszek, J. Cordeiro, and J. L. G. Dietz, Eds. SciTePress, Jul. 2013, pp. 257–264. [Online]. Available: http://dblp.uni-trier.de/db/conf/iceis/iceis2013-2.html#SilvaFJGO13 [retrieved: August, 2014]

[29] Svendsen, A. et al., "Developing a software product line for train control: A case study of cvl." in SPLC, ser. Lecture Notes in Computer Science, J. Bosch and J. Lee, Eds., vol. 6287. Springer, Sep. 2010, pp. 106–120. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-15579-6_8 [retrieved: August, 2014]

[30] Haber, A. et al., "First-class variability modeling in matlab/simulink," in Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems, ser. VaMoS '13. New York, NY, USA: ACM, Jan. 2013, pp. 1–8. [Online]. Available: http://doi.acm.org/10.1145/2430502.2430508 [retrieved: August, 2014]

[31] S. Blessing and M. Huhn, "Formal Safety Analysis and Verification in the Model Driven Development of a Pacemaker Product Line." in MBEES, H. Giese, M. Huhn, J. Phillips, and B. Schtz, Eds. fortiss GmbH, Mnchen, Feb. 2012, pp. 133–144. [Online]. Available: http://dblp.uni-trier.de/db/conf/mbees/mbees2012.html#BlessingH12 [retrieved: August, 2014]]

[32] M. Timmer, H. Brinksma, and M. I. A. Stoelinga, "Model-based testing," in Software and Systems Safety: Specification and Verification, ser. NATO Science for Peace and Security Series D: Information and Communication Security, M. Broy, C. Leuxner, and C. A. R. Hoare, Eds. Amsterdam: IOS Press, April 2011, vol. 30, pp. 1–32.

[33] Heimdahl, Mats Per Erik, "Safety and software intensive systems: Challenges old and new." in FOSE, L. C. Briand and A. L. Wolf, Eds., 2007, pp. 137–152.

[34] J. Liu, J. Dehlinger, and R. Lutz, "Safety analysis of software product lines using state-based modeling," in 16th IEEE International Symposium on Software Reliability Engineering, 2005. ISSRE 2005, pp. 10–30.

[35] H. Gomaa, Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Redwood City, CA, USA: Addison Wesley Longman, 2004.

[36] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, EMF: Eclipse Modeling Framework 2.0, 2nd ed. Addison-Wesley Professional, 2009.

[37] "Open source tool for UML modeling," 2011, http://www.papyrusuml.org/ [retrieved: August, 2014].

[38] SINTEF, "Cvl tool from sintef," 2012. [Online]. Available: http://www.omgwiki.org/variability/doku.php/doku.php?id=cvl_tool_from_sintef [retrieved: August, 2014]

[39] R. Prieto-Diaz and G. Arango, Domain Analysis and Software Systems Modeling. IEEE Press, 1991.

[40] C. W. Krueger, "Variation management for software production lines," in Proceedings of the Second International Conference on Software Product Lines, ser. SPLC 2. London, UK, UK: Springer-Verlag, Aug. 2002, pp. 37–48. [Online]. Available: http://dl.acm.org/citation.cfm?id=645882.672255 [retrieved: August, 2014]

 ISBN: 978-1-61208-367-4

Collaborative Team Management in Agile and Distributed Development Environments

NohSam Park and JongHyun Jang
IT Convergence Technology Research Laboratory
ETRI
Daejeon, Korea
{siru23, jangjh}@etri.re.kr

***Abstract*—The inherent nature of software engineering is collaboration. Recently software engineering practices have seen many agile methods, and distributed collaboration in geographically distant environment. In this paper, we propose the methods to manage the collaborative team for this changing environment. Collaborative team management skills in agile requires the communication skills and procedures in terms of social activities in agile process. In a distributed software project, human factors are emphasized for facilitating collaboration. The importance of risk management strategy is highlighted to address the circumstantial limitations of both environments. This paper presents the basic skills for an agile and distributed project, and reports on our experience of adapting for the real Studio project settings with the concrete methods.**

Keywords-Collaborative Team; Team Management; Agile; Distributed software development; Software Engineering.

I. INTRODUCTION

Software engineering is a result of team activity. Collaboration in software engineering has greatly increased thanks to widespread use of the Internet and many kinds of project management tools. Rapid development using agile methods also enabled various team organization and project management by emphasizing the communication process with customers [1].

Just like in many open source projects, distributed team formation may make communication more complicated because of time difference, culture, and language barriers. The wide range of engineers on the team may have different motivations and needs. These characteristics in global and diverse team management facilitate collaboration by offering technical tools and adaptive software processes. Teaming process research shows the importance of establishing and managing software teams and emphasizes the difficulties of implementing it [2]. Collaboration in software engineering refers to managing the entire lifecycle of the project, and it is the most important factor to accomplish high quality product, and efficient software engineering practices. Collaboration is complicated and hard to achieve because of the increased interdependencies between the project teams.

Agile software development has become popular since the early of 2000s, and involves collaboration and interactions naturally, resulting in creating working software [3]. The structure and organization of agile teams proves the people-focused approaches when it comes to collaboration.

The need for coordination in software project comes because tasks and artifacts between team members are tightly connected to each other, so researchers created a variety of tools and approaches to improve team coordination. In addition, some evaluation types and frameworks such as DESMET [4] for coordinating software engineering tools have been proposed [5].

Much work has been done in collaborative software engineering, but the collaborative practices are not routine and generalized. In a research field there are three main topics: theoretical understanding of collaborative software engineering, designing assessment methods for specific situations, and implementing tool support [1]. As should be clear from the practices and research work, collaboration is without doubt the core of software engineering. From the point of collaboration, it is required to develop the methods how to manage collaborative team in the current software engineering situations. As Austin and Devin described in their book [6], successfully managing knowledge workers – software team members – call for collaboration without detailed or coercive direction, keeping in mind that we cannot supervise talented employees in any conventional sense; we must lead them with passionate support and faith in their work.

This paper is organized as follows. In Section 2, some skills are proposed for enhancing collaboration in agile process and distributed development environment. Section 3 presents the case study of MSE Studio project at Carnegie Mellon University (CMU), and Section 4 concludes the paper.

II. COLLABORATIVE TEAM MANAGEMENT IN DIFFERENT ENVIRONMENT

Collaboration in software engineering has evolved through diverse processes, methodologies, and development environments. In this section, the ways of achieving the collaborative team management in the agile process and the distant development environment are discussed.

A. Agile Process Development Team

Customer collaboration and social activities get much emphasis in agile. Nevertheless, collaboration does not come naturally just by setting the agile team up. The team

 ISBN: 978-1-61208-367-4

management skills are important in order to improve collaboration and coordination, especially between the customer and software developers in the agile process.

Identify social skills for the agile process

The agile process has key practices such as small release, simple design, refactoring, and iteration. They also put an emphasis on communication with customers and reflection on development iterations. For example, pair programming in Extreme Programming (XP) [7] encompasses the whole communication not just involving two programmers in the same room. They discuss the problem, understand the task, negotiate their opinions and share the work.

Agile process practitioners need to socialize with coworkers and customers. Most of them are familiar with communicating using social networking and instant messaging. But, socializing also requires us to keep in contact with people in the physical situation. It involves respect for the difference, understanding people's situation, and sound critique towards participants. Socializing might cause conflicts among team members whether we apply it online or offline.

The team should identify diverse social skills from many different perspectives. From the technical view, the team encourages technical discussion and research. The team can have technical workshops or open-lab for intriguing the intellectual motivation. The working condition should easily accommodate the collaboration between people. Just like XP's pair programming requires the reconfiguration of desks, the working environment should be open and shared to increase collaboration.

Establish reporting channels between stakeholders

Co-located setting of agile processes does not require formal reporting procedures to keep managers and customers up-to-date with progress. Those procedures hinder the project from moving fast, which violates the agile property. The agile principle of "barely sufficient [8]," can be applied to reporting as well. The reporting concentrates on key features developed or requirements satisfied, removing any unnecessary information. But it should be able to hold the minimum value for the project.

Agile teams need to establish the reporting channel when they show the project progress information to the customer. Many agile teams still do in a light way such as spreadsheet, sticky notes on the wall or whiteboard. The intention was not try to impose additional burden or cost on the agile practices, but it is another option for the team that wants to use agile continuously.

Many tools can provide appropriate level of information for both managers and the customer. It would be the alternative for the formal reporting procedure between stakeholders in the agile methods. Plus, agile software tools provide reflections functionality when teams finish iteration for both developers and the customer. For example, by offering the burn-down chart, it shows the simple trend and increases understanding of the project progress.

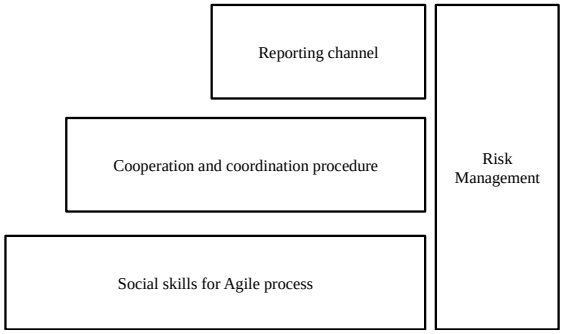


Figure 1. Collaboration skills for the agile process.

Establish a risk management strategy

The elements of the risk management paradigm are the following: *identify, analyze, plan, track, control, and communicate* risks [9]. Agile risk management follows the same activities like the traditional software projects. The iterative nature allows us to tackle high risk sooner than later. The risk management process is repeated every iteration, and remaining risks are re-assessed. Teams prioritize risks and take proactive risk management strategy for the top priority risks.

The pitfall of risk management in agile processes is that the team tends to dismiss the risks with low priorities when they assess the risks. People are likely to identify new risks for the project and focus on the high priority risks. In order to prevent the tem from overlooking those risks, the *risk overhaul* is suggested on every milestone of the project. Risk overhaul implies that existing risks are initialized and teams inspect risk management process from the scratch. From the risk identification to risk planning, teams go through every step involving the entire stakeholders. Teams can start with the remaining risks, and each risk is inspected thoroughly and reassessed.

In the risk overhaul, the outside member of the team can join with a fresh eye. In other words, every remaining risk should be treated and evaluated like newly identified risks. It could be burdensome and costly to do quite often, so it would be viable to perform it on a major milestone basis.

In Figure 1, collaboration skills for the agile process are described. Based on firm social skills, cooperation and coordination procedure should be established. On top of that, reporting channels enable the stakeholders to communicate effectively. Throughout these procedures, a proactive risk management should be implemented.

B. Distributed Development Team

The goal of distributed team building is to build a high performance team. Global Teaming goals are suggested in [10], each of which has specific practices and sub-practices when implementing a global software engineering (GSE) strategy. It has two specific goals: *Define Global Project Management, and Define Management between Locations*.

 ISBN: 978-1-61208-367-4

In distributed software development, diverse factors should be taken into account like distance, language, culture, etc. from the team setup. Especially, human factors are important for motivating participants and letting them take the initiative. The following are some suggestions for collaborative team management for distributed development teams.

Identify common goals, objectives as fast as possible

Distributed development settings require each team member to have consensus for the goals, objectives in the early phase of the project. But, the team members in different location have relatively fewer ways to get feedback and information for the project. They usually resort to online communications such as email, web-based tools, and social networking. Face-to-face interface like videoconferencing is possible, but still limited, especially when the team is globally distributed.

Distributed development teams should put much effort in getting all the stakeholders on the same page. The small problem in the early phase will snowball and end up bringing serious implications for the project. It is preferable to hold not only the kick-off meeting but also several workshops. Even though team members should be located in distance, it would be much better to get together.

Teams only work and collaborate when they share the same idea and goals. Though many technologies support meetings via audio or video, not all team members are comfortable because of diverse factors such as language barrier, time difference, etc. Just having a meeting does not guarantee to keep them agreed upon the issues. Follow-up activities should be implemented and the team should clarify the problem when they have issues during the meeting.

Define the explicit roles and responsibilities

Distant team should be given explicit roles and responsibilities for their team. Without them, the project manager will receive dozens of questions from distant development team members because they want to check what their missions and tasks are. The objective is to distribute work and motivate them to take the leadership of their own.

No one in the distant team would want to put his/her head up and lead without explicit roles and responsibilities. Make them take the initiative of the project, and make them feel they are the part of the team. When they can see what should be done throughout the project, they will make plan, accomplish tasks, and communicate as a whole team. The project manager should be able to inspire the distant team by setting the boundary of the central and distant team.

Partitioning and allocating tasks across the distant team is a key concept of the distant development. It is related to the team's capability to manage and develop features of the project. The project management should assess the distant team and local team's abilities objectively and modularize functional units.

Give autonomy and accountability

Some recommendations called "coherent and co-located teams of fully allocated engineers" were made for global software development projects [11]. They say that engineers should not be distracted by other tasks working on the same processes, methodologies, and terminology. The success of the distant development team comes from the innovation of the team members given autonomy.

The distant team can manage itself not by the central team's micromanagement. The distant team may have its own rules and management styles, thus it can make self-organized team. Then the central team gives it the necessary information, tools, and other resources in order to let it work. Product management would empower the distant team with the privilege and remove impediments in its way that may harm the progress of the project. All those things are related to promoting team performance in the project. The team as a whole can progress in its own roles and contribute to the project success.

The team needs to find the golden mean between autonomy and accountability. Autonomy should be allowed within the roles and responsibilities given by the central team. Autonomy and privileges should be only allowed in terms of the common goal: the success of the project. Autonomy naturally brings accountability for the team's result. Individuals in different locations work for the team and project's success, and each individual is responsible for their result. Use of different process can be done only when they meet the whole team's schedule, deliverables, and cost. The management should monitor and track team's progress and take actions to address the issues when autonomy gets on track of the project.

Relate the risks and problems

Distributed development projects bring additional high risk exposure as many risk factors exist such as culture-related and geographical-related risks. Bass et al. presented a coordination risk analysis method for multi-site projects in [12]. The team leader can start with this risk management strategy for the distant development project.

The team leader should relate the risks to the actual problems from these risks. It is the best to avoid risks or prevent them from becoming problems. But, some risks evade and become problems. In the risk management strategy, prioritizing and mitigating risks are highlighted, but not much attention is paid to the correlation between risks and problems when risks become problems.

Software engineers tend to either fix the problem or controlling the risk. We need to analyze the correlation between them, so that we can achieve more effective risk management. First, we analyze risk monitor, track and control activities. Then we look into what triggered the risk for becoming the problem, and what the problem's impact is. Investigating the reason and result of the problem helps us reflect on the risk management. That reflection keeps the risk with similar conditions from happening again. The risk/problem analysis process incorporates collaboration among physically distant team members.

 ISBN: 978-1-61208-367-4

III. CASE STUDY

In this section, we will give an example of CMU MSE (Master of Software Engineering) Studio project, and discuss issues when team management skill suggestions are applied to the real collaborative team setting.

A. The Studio project and team setup

MSE is a 16-month/4 semester intensive program for software engineers. The program can be done in the form of full or part-time via distance education as well. The entire program emphasizes application of course material in a hands-on experience with real, paying clients who expect actual deliverables [13]. CMU has been incorporating the core academics of software engineering into the MSE Studio.

The Studio project has three stakeholders: the team, mentors, and the client. The team is structured as a small with three to five students from diverse culture and backgrounds. Students are expected to overcome technical challenges, and meet their client's requests through the Studio project. Mentors are assigned to each team, and they conduct, advise and guide the project. Student-mentor meetings are held weekly in an interactive style of asking the student, encouraging reflections. The client requests the development of output by giving requirements and information, providing feedback, and evaluating the deliverables from the team.

Our team was composed of five team members, two mentors, and the client. Each team member is from different country. They speak different languages, and it means the team had various factors to consider such as language and culture. Work experience was also various from less than 1 year to more than 10 years. The client of the studio project came from the area of the retail store, which has many branches worldwide. The goal of the project was to improve the customer's shopping experience such as shortening the checkout time in the local store. In order to achieve the goal, the customer required us to develop a mobile application on the Android platform.

The team adopted OpenUp [14], which is one of the agile processes, as the development process. OpenUp has 4 phases of development lifecycle: *Inception, Elaboration, Construction, and Transition*. Though the team used the agile process, the client did not co-locate in the same place with the team. In the inception phase in OpenUp, the team was supposed to refine requirements and elicit specific features for the project.

The team had to take also another thing into consideration: one of the team members had to return to the home country and continue the academics in the transition phase.

B. Project Development and reflection about collaboration

1) Inception

In the inception phase of OpenUP, the team is supposed to establish the scope of project and do the requirements analysis. The team, however, did not get much response from the customer. The client stopped communicating once in a while and the team did not take the initiative meanwhile. We should have tried to fix the problem of miscommunication and come up with our own solutions despite of the client's absence. Basically we just waited the response from the client and we did not put much effort on the Studio project, which made the agile method ineffective. In addition, we did not prepare for the upcoming risk of the remote team setup.

2) Elaboration

In the elaboration phase, the tasks are mostly related to design. Architecture is believed to heavily affect the software and the team tried to convince the client to increase the communication for the architectural review. Technical risks were identified and reported to the client regularly, which made the team feel confident about the success of the project.

The team also established the project strategy during the elaboration phase not to repeat the mistake of the inception phase. That was mostly from what we learned in the architecture class, more specifically from Architecture-Centric Design Methodology (ACDM) [15]. We tailored the steps and procedures in accordance to our project context. The team also suggested the strategy and plan related to it. Contrary to the frustration in the elaboration phase, a well-established reporting channel and risk management strategy boosted the team morale as well as the client satisfaction.

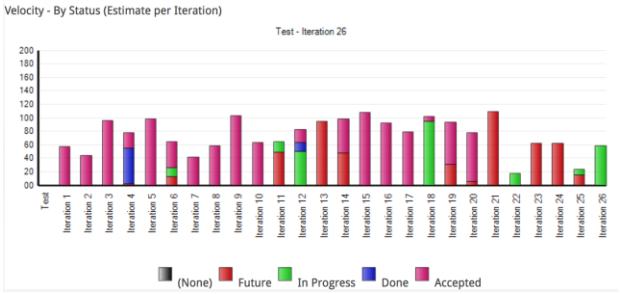
3) Construction

Implementing is what software engineers enjoy and indulge in the most. The team was given 48 hours of work each week. The common working time was set up during the weekdays to work together, and a daily scrum meeting was planned to check the status. With plenty time of work, developing two features the client asked was considered not a big deal for the team.

The plan was only a plan again, though. The chronic time management problem still did not show any hint of the improvement for some members. The daily meeting was switched to two meetings per week. The quality plan and the milestones have continuously changed because the client did not respond to any reports from us. It was the last opportunity for the team to co-work because one member would be in a remote place in the next phase. The Studio project is an academic course and that aspect heavily influenced for the team members. The benefits in the elaboration phase did not last long because the reporting channel with the client was collapsed and the social skills were useless.

The best lesson the team members learned is the importance of communication with the client. We were at a loss when the client just quit the connection and became contactless from time to time. This time, we changed the policy. The customer liaison, which has already existed, notified that he/she would try to contact with several times using email, text, and phone calls. When there was no response for those efforts, the team finally made their own decisions. At least, we tried to remove some uncertainties and the team was able to deliver the mobile application

 ISBN: 978-1-61208-367-4



(b) Velocity Status

Figure 2. Team's velocity trend

integrating two features. That would not be exactly what the client wanted at first, but it was the minimum we could make without enough communication.

4) Transition

The team finally faced the distant team setting in the transition phase. Actually, that intrigued the team because it was not common in small and medium sized businesses. The transition phase normally does not involve many tasks related to the development, so it was not easy to assess how the suggestions for the remote collaboration would work.

Collaborating as a team can be a real challenge. Getting everybody on the same page, assigning tasks, following up on pending items, and making sure everyone is always in the loop is never easy, and it is something almost all companies struggle with. The team decided to have a weekly meeting considering the time difference and team members' schedule. The team used a videoconferencing tool like Skype or Hangout of Google to get together. The team prepared the remote development condition from the construction phase, but that was not enough. More documents were needed for the remote member to catch up. More methods to collaborate online should have been attempted.

In the early weeks of the transition phase, the weekly meetings were canceled or held without getting the whole team members. The meeting itself was not satisfactory: just checking and reporting the status without enough discussion and review of the deliverables and the iteration process. The team did not take advantage of the current collaboration technologies. Whereas the Studio project did not see the effectiveness from the remote team condition, in another situation of the remote class we took at the same time, the collaboration was good enough. The class asked for the group presentation about one topic and we were in the same distant team setting. The group shared the goals of the presentation and divided the parts each member had to do. We had a weekly meeting to check each member's progress after working individually. A subjective criterion would be the members' morale whereas an objective one would be the grade for each class. The results in the Studio project were poor in both criteria.

One way of assessing the success of the team in the agile method is the trend of team's velocity. It could be applied to the evaluation of distant team in the agile method. Comparing the velocity in the co-located situation with one in the remote condition will show the effectiveness of the team's status. The team's velocity did not show the improvement during the project in Figure 2. Overall the trend is not stable except during the elaboration phase from iteration 12 to iteration 17. Some tasks are not finished on time during the iteration in the transition phase after iteration 22.

C. Discussion

In this section, we will investigate how these suggestions would make better this situation or what were the issues when adopting these into the real situation.

1) Agile team

Identifying social skills refers to acquiring diverse communication methods both among team members and for the customer. Even though the agile method was adopted, which requires the intimate and quite often conversation, the team was too passive to just wait requirements from the customer. The team established several kinds of communication methods: a *Facebook group* between team members in addition to traditional ways such as email and instant messaging, and *biweekly teleconference meeting* with the customer.

Reporting channels are the official procedure for discussing, negotiating and satisfying the customer expectation for the project. The team did not have the on-site customer even though adopting the OpenUp. So the team needed to set up the reporting channel, and a biweekly teleconference meeting was held with the customer to report the progress of the project, and the customer gave feedback about it.

Risk management strategy is emphasized by the nature of the OpenUp requiring risk management process at the end of iteration. The team adopted the aforementioned risk overhaul in the elaboration phase. The customer wanted the team to follow feature-by-feature development for the mobile application. When we touched another one after finishing one feature, new kinds of risks were identified and the team needed to see it differently from the usual risk management process.

As these suggestions were applied to the real agile project, the problem behind them is always the motivation. When the team members are not motivated to use them, the collaboration skills are meaningless. In fact, the team was not able to build some management foundation before realizing the team's collaboration and coordination problems and raising the awareness of the importance of them.

2) Distant team

Sharing common goals and vision in the early phase of the project is the first thing we had to consider. The team had co-located setting except in the transition phase that was good enough for having the common goals and objectives. Maintaining the commonality, however, should be kept throughout the entire project when the change happens.

 ISBN: 978-1-61208-367-4

TABLE I
COLLABORATIVE SKILLS FOR THE STUDIO PROJECT

Skills	Team's methods	Criteria for the skills
Social skills	Email, Facebook group, biweekly teleconference meeting	Team members' and client's morale (questionnaire)
Reporting channels	VersionOne report, biweekly teleconference meeting	Number of reporting
Risk management strategy	Risk evaluation at the end of iteration Risk overhaul	Trend of the number of risks
Common goals in the early phase	Requirement engineering (RE) in co-located environment	Time spent in RE Number of requirements
Explicit roles and responsibilities	Assign of role to each member	Assigned roles
Autonomy and accountability	Distant team member management by formal (VersionOne) and informal (regular videoconferencing) method	Progress report by team member
Relate risk to problem	Risk/problem analysis	Number of problems from the risks

Explicit roles and responsibilities are a factor which enables to proceed in the distant development environment. The distant team member should be able to know what his tasks are, when they should be done, and how they can be incorporated into the deliverables of the project. It is only possible when the team defines roles and responsibilities for each team member.

Autonomy and accountability is an integral part when we deal with the team morale and the project accomplishments. In reality, it is not feasible to micromanage the distant team member. One of solutions is to give autonomy and ask accountability for the results. The team leader or project management should be able to ask for accountability for his tasks.

Risk/problem analysis is supplemental to the existing risk management process. A risk may become a problem or not, and the distant team condition may bring confliction when it becomes a problem. Without complete analysis about the reason and implications of the risk/problem, the team might evade the responsibility or accuse someone else who is not present, thus infringing collaborative team spirit.

Table 1 summarizes the suggested skills and the corresponding methods in the Studio project. The criteria for the skills are measured by both subjectively like questionnaire and quantitatively.

IV. CONCLUSION

In this paper, the trend of collaboration in software engineering was reviewed, and some suggestions were proposed for the agile process and distributed development environment. Agile process is known for strengthening the collaboration with the customer, but it is necessary to prepare strategy and procedure beforehand about how to communicate both within the teams and among the customer.

Management skills in distributed development environment presented in this paper focuses on human factors. Respecting, understanding given circumstances of each team will facilitate the collaboration. Besides, thorough preparation and planning regarding how to manage the project will drive collaborative team members to follow the practices of software engineering.

Some issues and reflections are discussed when we implemented these skills into the real software project. Our team had both characteristics of agile and distributed development. We learned that coordinating and collaborating are hard to obtain from some of experience in the project because of human and technological factors.

ACKNOWLEDGMENT

This work was supported by Electronics and Telecommunications Research Institute (ETRI) Grant funded by the Korea government [14ZC1320, Development of WoT Collaboration Service Platform based on Social Relation].

REFERENCES

[1] Ivan Mistrik, John Grundy, Andre van der Hoek, and Jim Whitehead, Collaborative Software Engineering. Springer, 2010.

[2] Ita Richardson, Valentine Caseyb, Fergal McCafferyb, John Burtonc, and Sarah Beechama, "A Process Framework for Global Software Engineering Teams," Information and Software Technology," vol. 54, pp. 1175-1191, November 2012.

[3] Helen Sharp and Hugh Robinson, "Collaboration and coor-dination in mature eXtreme programming teams," International Journal of Human-Computer Studies," vol. 66, pp. 506-518, July 2008.

[4] Barbara Kitchenham, Stephen Linkman, and David Law, "DESMET: a methodology for evaluating software engineering methods and tools," Computing & Control Engineering Journal, vol. 8,pp. 120-126, June 1997.

[5] Barbara Ann Kitchenham, "Evaluating Software Engineering Methods and Tool Part 1: The Evaluation Context and Evaluation Methods," SIGSOFT Software Engineering Notes, vol. 21, pp. 11-14, January 1996.

[6] Rob Austin and Lee Devin, Artful making: What Managers Need to Know About How Artists Work, Prentice Hall, 2003.

[7] Kent Beck and Cynthia Andres, Extreme Programming Explained, Addison-Wesley, Reading MA, 2000.

[8] Kevin Tate, Sustainable Software Development: An Agile Perspective, Addison-Wesley Professional, 2005.

[9] Ray C. Williams, George Pandelios, and Sandra Behrens, Software Risk Evaluation (SRE) Method Description (Version 2.0), SEI, 1999.

[10] Ita Richardson, Miriam Q'Riordan, and Valentine Casey, "Knowledge Management in the Global Software Engineering Environment," ICGSE 2009, pp. 367-369, 2009.

[11] Christof Ebert and Philip De Neve, "Surviving global software development," IEEE Software, vol. 18, pp. 62-69, March 2001.

[12] Matthew Bass, James D. Herbsleb, and Christian Lescher, "A Coordination Risk Analysis Method for Multi-Site Projects:Experience Report," 2009 IEEE International Conference on Global Software Engineering, pp. 31-40, November 2009.

[13] Mary Shaw, Jim Herbsleb, Ipek Ozkaya, and Dave Root, "Deciding What to Design: Closing a Gap in Software Engineering Education," ICSE 2005, pp.607-608, May 2005.

[14] Eclipse Foundation, OpenUp Wiki, http://epf.eclipse.org/wikis/openup, 2014.08.15.

[15] Anthony J. Lattanze, Architecting Software Intensive Systems, Auerbach Publications, 2009.

 ISBN: 978-1-61208-367-4

Security Through Software Rejuvenation

Chen-Yu Lee, Krishna M. Kavi, Mahadevan Gomathisankaran, Patrick Kamongi

Department of Computer Science and Engineering
University of North Texas
Email: {Chen-Yu.lee,Krishna.Kavi, Mahadevan.Gomathisankaran}@unt.edu
Email: patrickkamongi@my.unt.edu

***Abstract*—Software rejuvenation has been used to improve reliability of systems by periodically checkpointing and restarting them. In this paper, we propose to use rejuvenation as a mechanism to enhance the security of Cloud infrastructure and eliminate malware by continuous and periodic rejuvenation. To evaluate the effectiveness of rejuvenation in eliminating malware, we defined an experimental setup, and utilizing complete system rejuvenation, as well as application level rejuvenation we investigated which malware were eliminated. We also describe a cost model for rejuvenation so that one can determine how often systems and applications should be rejuvenated, trading cost against security. Our experiments and models show that rejuvenation once every 24 hours is cost-effective.**

Keywords–Rejuvenation; Malware; Security; Vulnerability.

I. INTRODUCTION

Computer viruses have been evolving into more complex malware and the detection and elimination of such threats is becoming very expensive in large IT operations. The number of new types of malware detected over the past ten years has increased very rapidly since 2010.

Software Rejuvenation technology was first proposed by Lin in 1993 [1]. The author observed that system performance degrades with time, and failure rates also increase with time. This phenomenon was termed software *aging*. A proactive solution to this problem is to gracefully terminate an application or a system and restart it in a clean internal state, known as *software rejuvenation* [2]. Rejuvenation technology was originally used for software fault tolerance [3] [4]. The most relevant work that applies rejuvenation for protection against security attacks is SWRMS proposed by Aung in 2004 [5]. The authors propose to identify attacks using an intrusion detection system, and then perform software rejuvenation to counteract these attacks, including killing the intruders' processes, halting abuse, shutting down unauthorized connections, and restarting applications. The attacks, however, are not eliminated if the processes are infected. They do not rely on rollback to restart infected processes from a known clean state. Moreover, since the approach is based on detecting intrusions, one should include the cost of detecting attacks along with the cost of rejuvenation, to estimate the total cost of their approach. We do not base rejuvenation on detecting attacks; rejuvenation is applied regularly. Along with rejuvenation, we restart processes from a checkpointed or clean state.

More generally, we believe that rejuvenation can either be used in place of scanning to detect attacks and malware, or in addition to scanning. To evaluate the effectiveness of rejuvenation against malware and viruses, we created a testbed that performs both system level and application level checkpointing and restarting. We then introduced known malware and verified if the malware was eliminated after the restart. The testbed also provides for a realistic evaluation of the cost of rejuvenation and which malware can be eliminated using rejuvenation. In this paper, we also develop a model to compare the cost of rejuvenation with the cost of scanning for malware. We emphasize that rejuvenation is more than just restarting of systems, it also includes checkpointing software applications and systems in clean states, and periodically rolling back the software to known clean states.

The rest of the paper is organized as follows: Section II introduces how rejuvenation can be used to enhance security. This section also introduces a model for estimating the cost of rejuvenation. Section III shows the simulations results of rejuvenation on our web service built by Joomla [6] and OpenStack [7]. Section IV compares rejuvenation with scanning approaches. Section V provides our conclusions and future work.

II. REJUVENATION FOR SECURITY

A. Environment Description

The proposed rejuvenation is applied to Cloud computing [8] environments to enhance security and stability of the systems. Many commercial operations rely on Cloud computing and in such applications, maintaining low Mean Time To Repair (MTTR) and the cost of repair are essential to the profitability of the operations. Therefore, they normally use software patches to fix problems, instead of completely overhauling their systems. The patches include system patches, software patches, malware/virus signatures, firewall rules, etc.

B. Work Flow of Rejuvenation for Security

We propose to use periodic rejuvenation (i.e.,checkpoint, rollback, recover and restart) to improve security and reliability of components. The rejuvenation can be applied modularly to minimize the downtime of the system. Each module is restored (or rejuvenated) to a clean checkpoint and reconnected with other related modules. Patches can also be applied to modules during a rejuvenation to reduce their vulnerabilities and to eliminate detected malware. The patches should be verified as clean and distributed by authorized providers, to assure that patched modules are clean. The work flows are shown in Figure 1, and the main processes are described below:

 ISBN: 978-1-61208-367-4

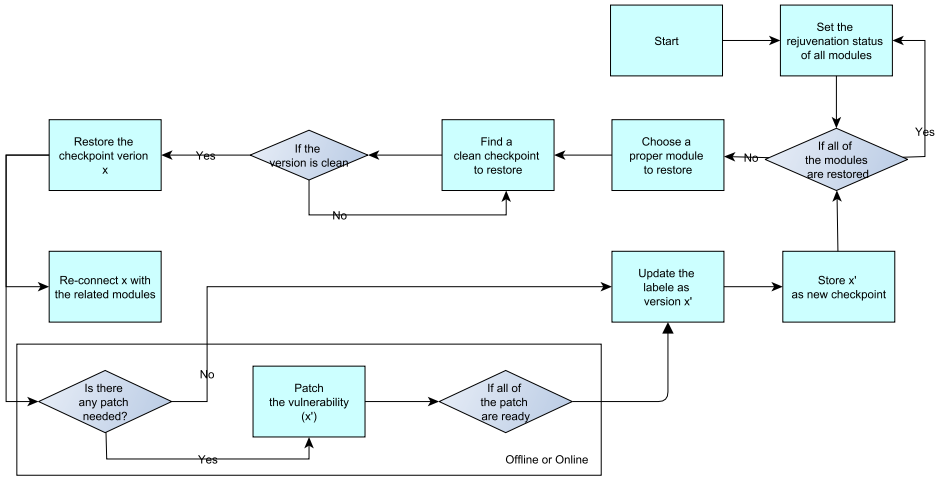


Figure 1. The workflow of secure rejuvenation mechanism

- Checkpoint: When a new software module is tested, verified, and ready to go online, it is assumed to be in a clean state and a checkpoint of the module is taken. Periodically, the module is rolled back to the clean checkpoint to scrub the module of any infections. If any design fixes or other patches are made available to the module since its original release (and the patches are verified as trusted and clean), the module is upgraded during the rejuvenation period, and the checkpoint image is also updated to the new clean state.
- Recover: All modules of a system go through a rejuvenation process (checkpoint-recovery) periodically, where the periodicity is determined based on the cost of rejuvenation and the frequency of new malware introductions. The process eliminates not only software aging and soft or intermittent faults, but also some malware. The rejuvenation may also be performed when an abnormal condition or a suspected security threat is detected.
- Restart: The module always restarts after each recovery. This eliminates software aging and some common security threats, including denial of service (DoS) and others.

III. Simulation Results

A. Simulation Settings

To understand how rejuvenation can eliminate malware, we built a Joomla content management service on our private Cloud environment supported by OpenStack [7]. The specification of our system is shown in Table I.

B. Results and Analysis

The rejuvenation is divided into two types: a complete system rejuvenation and component rejuvenation. In our simulation, the complete system rejuvenation is provided by OpenStack which creates an instance snapshot stored in the snapshot repository, and restores it while launching the complete rejuvenation. It supports live snapshotting, which allows for taking snapshots of the running virtual machines without pausing them. In the second case, we only take checkpoints of an application or components of applications and periodically restore them.

In the experiment, we tested some malware and vulnerabilities listed in Table II. Most of the attacks are at the Operating Systems (OS) level: they create backdoors for bot or other attacks by using rootkit or other related technologies. Under the attack, the malware gains root privileges, hides itself, and deletes itself from the log. We perform our experiment in two phases.

Phase 1. Inject malware and scan the complete system to make sure the malware is in the system and detectable by anti-malware software. NOD32 [9] and ClamAV [10] are applied in our experiments.

Phase 2. Restore the checkpointed version and then scan to find if the malware is eliminated by rejuvenation. If the malware is not eliminated by the rejuvenation, it should be detected by anti-malware software.

The experimental result shows that the complete system rejuvenation eliminates all the malware we introduced and recovers all of the infected files. We also simulated changes to the integrity of MySQL [11] files using the known vulnerabilities of MySQL. After restoring, the modified files are recovered correctly. The rejuvenation can only recover the

TABLE I. textscSimulation Environment Specification

Platform	Version
Cloud platform	OpenStack
Flavor	m1.small
RAM	2GB
Processor	QEMU Virtual 1.0@2.33GHz(1Core)
Instance operation system	Ubuntu 12.04
Instance Size	20 GB
Application service	Joomla 3.3
Database	MySQL 5.5.36
Compiler	PHP 5.4.27
Web service	Apache 2.4.9
Anti-malware software	ClamAV 0.98.3
Anti-malware software	F-prot 6
Anti-malware software	Nod32 4

 ISBN: 978-1-61208-367-4

infected files, but the vulnerabilities still exist making the system vulnerable for repeated attacks. Vulnerabilities can only be fixed with appropriate patches. We also tested to show that rejuvenation can rescue the system from Denial of Service (DoS) [12] or low-rate DoS attacks [13].

TABLE II. TEST MALWARE AND VULNERABILITIES

Malware	Scope	Rejuvenation, Result
Backdoor.Linux.Ovason	Operation System	Restore, Eliminated
Backdoor.Linux.Phobi.l	Operation System	Restore, Eliminated
Backdoor.Linux.Rst.a	Operation System	Restore, Eliminated
Exploit.Linux.Da2.a	Operation System	Restore, Eliminated
Exploit.Linux.Race.l	Operation System	Restore, Eliminated
Net-Worm.Linux.Scalper.b	Operation System	Restore, Eliminated
Rootkit.Linux.Agent.sm	Operation System	Restore, Eliminated
Trojan.Linux.Rootkit.n	Operation System	Restore, Eliminated
Trojan.Tsunami.B	Operation System	Restore, Eliminated
VirTool.Linux.Mhttpd	Operation System	Restore, Eliminated
Virus.Linux.Osf.8759	Operation System	Restore, Eliminated
Virus.Linux.Radix	Operation System	Restore, Eliminated
Virus.Linux.Silvio.b	Operation System	Restore, Eliminated
Virus.Linux.Snoopy.c	Operation System	Restore, Eliminated
Vulnerability	**Scope**	**Rejuvenation, Result**
CVE-2013-1636	Joomla	Restore, Recovered
CVE-2014-2440	MySQL	Restore, Recovered
CVE-2014-2436	MySQL	Restore, Recovered
Attack	**Scope**	**Result**
Denial of Service (DoS)	Apache	Reboot, Recovered
Low-rate Dos	Apache	Reboot, Recovered

C. Performance and Cost

Performance of rejuvenation (the time spent for rejuvenation) is important because it relates to the unavailability or downtime of the system or service. In this section, we report the performance of our simulations in two parts: time spent for rejuvenation and the storage space required for storing checkpointed information.

1) Time spent and storage space costs: For the complete rejuvenation experiment, we checkpointed and restored the working instance. But, checkpointing can be performed without any downtime; restoring, however causes 54 seconds of downtime. For component rejuvenation, we set up a checkpoint on Apache and MySQL database software in our experiments with 850 MB of data without pausing; the chekpoint images used 327MB of memory. Application level rejuvenation required 187.51 seconds (includes the time to retrieve checkpointed images, stopping the application and restarting the application using the checkpointed image).

2) Cost: Globalscape found that in 60% of Fortune 500 companies, a single hour without critical systems costs their company between $250,000 and $500,000 and one in six companies reported that one hour of downtime can cost $1 million or more [14].

Assuming that on average $500,000 of loss per hour of downtime, our experiments show a cost of $7500 for complete system rejuvenation, $26,043 for rejuvenating Apache and MySQL software. The cost of the storage needed for checkpoint images are $2.4 and $0.12 respectively based on Amazon's prices. The cost model is described in Section IV-B.

IV. ANALYSIS AND COMPARISON OF REJUVENATION FOR SECURITY

In this section, we compare the capabilities in term of defense against various security threats and cost associated with rejuvenation and malware scanning techniques.

A. Characteristics Comparison

Rejuvenation has been used as a fault-tolerant/fault-avoidance approach in software systems. In a similar manner, rejuvenation can be applied as a defense against security threats. By restoring components to clean or healthy states, rejuvenation can make the system less prone to catastrophic failures. In Table III, we compare the capabilities of rejuvenation with malware scanning when applied to survivable systems. In Section III, our experiments have shown that rejuvenation can eliminate or mitigate the effects of several types of malware. Some weaknesses that cannot be eliminated using rejuvenation include trapdoors, which are eliminated by compiler-based code checkers and detected by resource monitors. But, anti-malware software need to monitor and scan entire memory and file systems to detect malware and eliminate or quarantine the infected files.

TABLE III. THE COMPARISON OF THE FEATURES, AND THE ABILITIES OF THREAT ELIMINATION BETWEEN REJUVENATION AND MALWARE SCANNING FOR SECURITY.

Feature	Rejuvenation	Malware Scanning
Fault avoidance	Partial	No
Fault tolerance	Yes	Yes
Denial of Service(DoS) or Low-rate DoS	Reboot	Log analysis
Virus elimination	Restore to checkpoint	Scanning
Trojan horse elimination	Restore to checkpoint	Scanning
Trapdoors elimination	No	No
Automated software-patching	Yes	Yes
Intrusion dection	No	Yes

B. Cost Model

This section discusses the cost of performing rejuvenation compared with malware scanning more formally.

Malware scanning software (e.g., anti-malware software) is usually performed as a daemon, scanning all the stored files, executing processes, system kernel and other system software continuously. Scanning may detect more security threats than that can be eliminated using only rejuvenation. However, scanning for malware consumes computational resources and thus the following model can be used for estimating the cost of malware scanning ($CoMS$).

- Instance size(V): The cost of scanning is proportional to the size of the system being scanned. In addition to scanning of the system at startup, malware scanning occurs continuously and is invoked when changes to the system are detected (such as file updates, internet downloads, mail attachments or other changes to the system state, such as changes to page tables). In this paper, we relate the cost of scanning to the average volume of the new information that must be scanned over a given period of time. The period of time and the volume of data scanned are compared with the rejuvenation period and the volume of information involved in the rejuvenation process.
- Scan speed(SS): This is the rate at which a system can be scanned to detect malware or virus signatures.
- Cloud computing fee (CCF): The fee charged by Cloud providers (whether the computing is used for scanning or for providing services).

 ISBN: 978-1-61208-367-4

The total cost involved with malware scanning C_{MS} for size V over a chosen time period T is

$$C_{MS}(V,T) = \frac{V \times CCF}{SS} \quad (1)$$

As an example, if it is assumed that the scanning speed is 26.58 MB/sec [15] and the computing fee charged by Amazon EC2 is \$0.176, \$0.351, \$0.702, and \$1.404 per hour for instance size 4, 32, 40, and 80GB [16], the cost of performing one malware scan on a Cloud environment with 10 GB size data would be \$0.007, \$0.117, \$0.293, and \$1.173 respectively.

In this paper, we assume two types of rejuvenation: a regular, periodic rejuvenation at fixed periods, and ad hoc rejuvenations when anomalies or threats are detected. Thus, factors that contribute to the cost of rejuvenation are divided into two parts. One is the cost involved with rejuvenation (CoR), and the other is involved with monitoring (CoM) to detect anomalies. Rejuvenation makes some modules unavailable (downtime) during the process of restoration. The following are the factors that influence the cost of rejuvenation.

- Downtime (DT): While performing the rejuvenation, some modules will be unavailable and the downtime can range from a few seconds to a few minutes.
- Number of transactions lost (TL): The number of transactions lost during the downtime.
- Potential loss of revenue associated with each transaction (PR) that could not be completed during downtime.
- Version storage fee (SF): Since clean modules and checkpointed states must be saved, we include the cost of storage with rejuvenation. In some cases, we may need to save m snapshots or checkpoints to fully recover the system to a clean state. Thus we include the total cost of storage needed for checkpointing. This can be compared with the volume scanned by malware scanners.
- Data transfer fee (TF): We assume that the checkpoints are stored in a backup or archival facility and this information has to be transferred to executing environments during restoration. We include the data transfer costs for transferring n bytes of data transferred between an execution environment and backup facility.

The total cost involved with rejuvenation $CoR_{periodic}$ for size V, with a rejuvenation period of T, is

$$CoR_{periodic}(V,T) = DT \times TL \times PR + mV \times SF + V \times TF \quad (2)$$

In addition to periodic rejuvenation, ad hoc rejuvenation (rollback to clean checkpoint and restart) is also applied when an abnormal condition or a security violation is detected. The detection may be based on monitoring system performance or other indicators. For example, performance indicators, including memory allocations, CPU usage, network traffic, disk writes, may indicate abnormal behavior of applications. We will include the cost of monitoring ($CoM(V,t)$) the system to identify abnormal conditions in the cost of rejuvenation. The cost depends on the volume V of information monitored.

$$CoR_{adhoc}(V,t) = CoM(V,t) + CoR_{periodic}(V,t) \quad (3)$$

Since the ad hoc rejuvenation can take place at any time between scheduled periodic rejuvenations, we will use a probability distribution that associates the probability of detecting an abnormal condition over this period of time. We can now compute the expected cost of rejuvenation that includes both ad hoc and periodic rejuvenation as follows.

$$CoR_{total}(V,T) = \int_0^T f(t)(CoM(V,t) + CoR_{periodic}(V,t))\,\mathrm{d}t \quad (4)$$

Here, $f(t)$ is the probability density function that reflects the probabilities of detecting abnormal behaviors. $f(t)$ varies depending on the security environment of the institution. If the systems are not protected, the probability of detecting an abnormal system may be higher. T is the scheduled rejuvenation period.

Consider for example that it takes 17 seconds to rejuvenate a system (i.e., the downtime is 17 seconds [17]), the average number of transactions lost in a year is 355.72 [17], the average potential revenue of a transaction lost is \$100,000, and the storage fee charged by Amazon is \$0.095 per GB-month and data transfer fee is \$0.12 per GB, the cost of performing each periodic rejuvenation is \$19.1756 for the 10 GB cloud instance. If we assume that in addition to hourly scheduled rejuvenation, ad hoc rejuvenations are warranted with a probability of 10% in between scheduled rejuvenations and if we assume that monitoring consumes 0.1% of CPU time, the total cost of rejuvenation can be estimated as

$$CoR_{total}(10GB, 1hr) = 2.03756 + 19.1756 \quad (5)$$

C. Cost Comparison

Figure 2 shows the cost of rejuvenation performed periodically for different frequencies: four hours, six hours, 12 hours, and 24 hours over a year. The cost of rejuvenation over a year depends on the frequency of rejuvenation and the cost of each rejuvenation. In Figure 2, we did not include monitoring and ad hoc rejuvenation costs, since these costs depend on the probability of detecting an abnormal condition. The figure also shows the cost of scanning for malware. The cost of scanning depends on the size of the system being scanned. The red horizontal lines represent the cost of scanning continuously ($Frequency = 0$). We also show the cost of malware scanning when scanning takes place at four, six, 12 and 24 hour periods - similar to rejuvenation. Systems need only to scan new information generated during the period and we assume that the amount of new information generated is proportional to the length of the period. It can be seen that the cost of rejuvenation decreases with the decrease in frequency (less frequent rejuvenation). The cost of continuously scanning for malware (see the three dash lines) is higher than rejuvenation at certain rejuvenation periodicities. If one assumes that

 ISBN: 978-1-61208-367-4

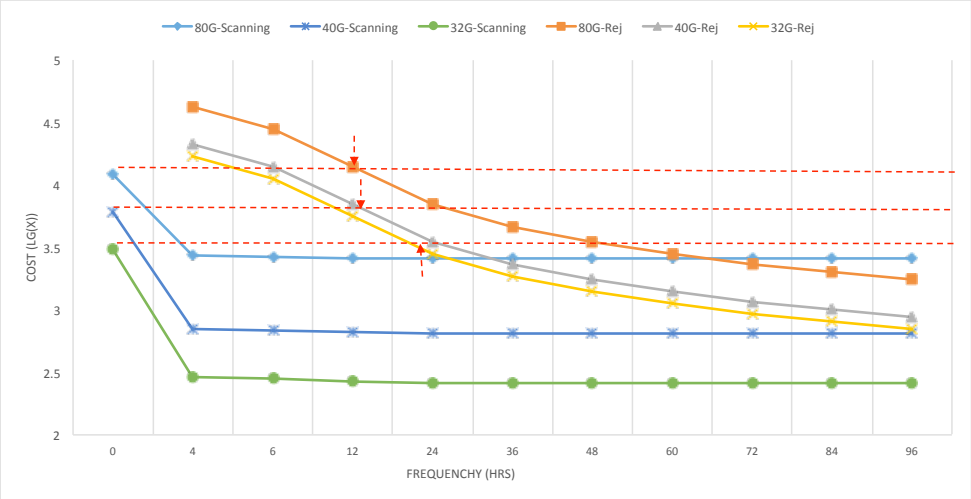


Figure 2. Cost comparison of secure rejuvenation versus malware scanning

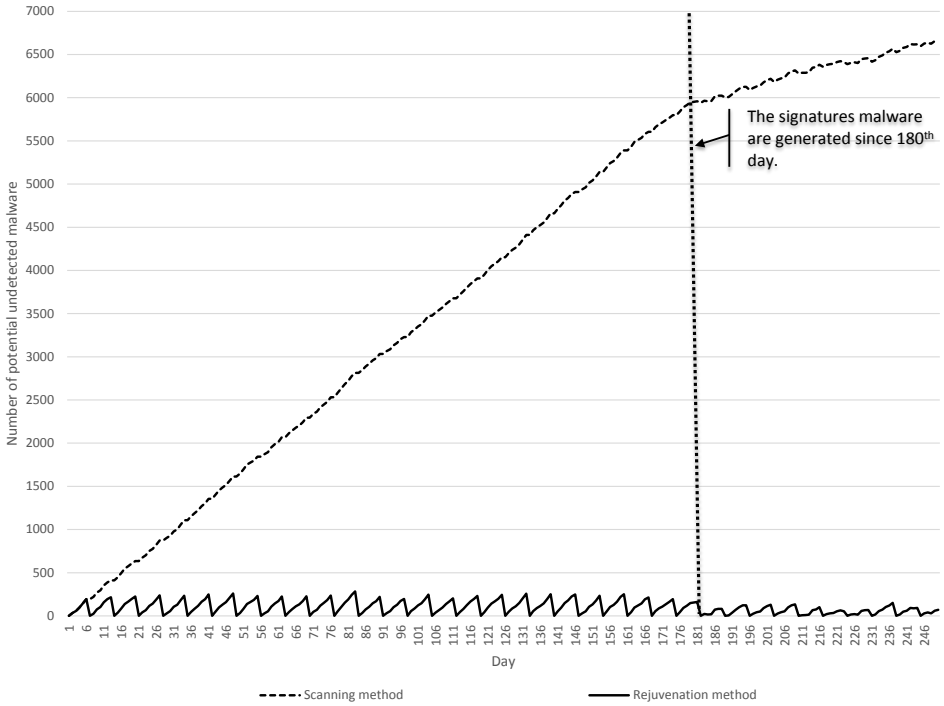


Figure 3. The potential number of malware remaining in a system after use of scanning versus rejuvenation

systems scan for malware at fixed internals (such as every four hours), rejuvenation costs are higher except when it takes place once every 80-100 hours. Based on our assumptions and cost models, rejuvenation once every 24 hours appears to be a reasonable choice for different system sizes.

D. Undetectable Malware Elimination

Malware is getting more sophisticated and the sophistication is increasing in recent years. McAfee's report shows that there are over 100,000 new malware instances detected in a given day [18]. There are three phases in the detection and elimination of malware. The first is the undetected phase in which the malware strain was not detected in the system. The second is the identification phase in which the malware strain is detected as a malicious code pattern and its signature is generated. Finally, the malware strain enters the detected phase after its signature is updated. A study by Damballa demonstrated that the typical gap between malware release and detection using anti-malware is 54 days, almost 8 weeks [19]. Nearly half of the 100,000 malware samples go undetected on the first testing day, and there were at least 15% of the samples remaining undetected even after 180 days. This means that the system may suffer from undetected malware for long periods of time.

 ISBN: 978-1-61208-367-4

Suppose a system component is infected with an average of 30 malware strains every day since it is released, the number of potential malware strains hidden may increase over the next several weeks before some strains are detected. On average it will be 9 weeks before detected malware signatures are released, and the number of hidden malware will be reduced as shown in Figure 3.

By contrast, the proposed rejuvenation mechanism periodically restores the component to a "clean" version (checkpoint); thus, the exposure of the system to new malware introduction is the time between rejuvenations. Assuming that the component is rejuvenated once a day, it remains in "clean" status at the beginning of each day. After the 9th week, some malware strains are eliminated because of the signatures, thus the potential malware strains lurking may decrease as long as the backup version is not infected.

E. Complete Rejuvenation and Component Rejuvenation

The services can be rejuvenated one service at a time such that the impact of rejuvenation is not felt by the entire system. The performance of rejuvenation depends on the instance's capability. By taking our experiment as an example, the instance works with flavor m1.small, thus the ability of checkpointing smaller size files is slower than checkpointing of complete instance performed by the host. Furthermore, any patches or upgrades to services can be done separately from a running system.

F. Application of Mobile Device

Kaspersky Lab's report shows that approximately 10,000,000 unique mobile malicious installation packages were detected in 2012-2013 [20]. Sometimes mobile malware resists the anti-malware protection because of Android vulnerabilities. Malware uses the vulnerabilities to bypass the code check, enhance the privilege to extend their capabilities, and make it more difficult to be removed, like Trojan-SMS.AndroidOS.Svpeng.a. Therefore, it is difficult for normal users to remove malware, since most of the malware is embedded in the legitimate software and acquires administrator privilege during the installation. There are only two options for users. One is to reset the system to factory settings, but some malware could obstruct this reset. The other is to apply anti-malware software to continuously scan, analyze, and eliminate it; but this consumes processing and thus the battery life of the device.

Our rejuvenation mechanism can be applied on Virtual Machine-based environments, such as cloud services, as well as mobile devices (e.g., Android). Our rejuvenation mechanism restores the checkpointed image from either the storage of the device or from some external storage in the Cloud, or may rely on trusted zones to bring the system to a clean or consistent state. If the rejuvenation is performed while the device is connected to a power source, the battery life will not be a consideration. Rejuvenation can be performed on a regular basis, similar to checking periodically for software patches and upgrades. A rejuvenation mechanism, therefore, is more suitable in a mobile environment, than malware scanning techniques.

V. Conclusion

The cybersecurity of Cloud-based computing systems are becoming critical to modern society as we are becoming ever more dependent on information infrastructures. Balancing system reliability, availability and security is complex. Malware and other security threats are becoming more sophisticated. Thus a multipronged approach is necessary to improve security as well as system survivability. We feel that software rejuvenation, which has been successfully employed as a fault-tolerance mechanism, can also be used as a defense against security threats. We conducted experiments in a controlled environment to show that rejuvenation does eliminate some malware. We will extend our experiments to more thoroughly evaluate which types of malware can and cannot be eliminated with rejuvenation only. In this paper, we also introduced a model that can be used to compare the costs associated with rejuvenation and malware scanning so that one can determine the rejuvenation frequencies that lead to cost-effective defense against hidden threats. While we compared rejuvenation as an alternative to scanning in this paper, they should be used together.

Acknowledgment

This research is supported in part by the NSF Net-centric and Cloud Software and Systems Industry/University Cooperative Research Center and NSF award 1128344 and 1332035. The authors also acknowledge David Struble's help in making the paper more readable.

References

[1] F. Lin, "Re-engineering option analysis for managing software rejuvenation," Information and Software Technology, vol. 35, no. 8, 1993, pp. 462–467.

[2] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software rejuvenation: analysis, module and applications," in Proceedings of the 25th International Symposium on Fault-Tolerant Computing (FTCS-25), 1995, pp. 381–390.

[3] R. Agepati, N. Gundala, and S. V. Amari, "Optimal software rejuvenation policies," in Proceedings of the Reliability and Maintainability Symposium (RAMS), 2013, pp. 1–7.

[4] S. Oikawa, "Independent Kernel/Process Checkpointing on Non-Volatile Main Memory for Quick Kernel Rejuvenation," in Proceedings of the 27th International Conference on Architecture of Computing Systems (ARCS), ser. LNCS. Springer, 2014, vol. 8350, pp. 233–244.

[5] K. M. M. Aung and J. S. Park, "Software Rejuvenation Approach to Security Engineering," in Proceedings of the International Conference on Computational Science and Its Applications (ICCSA), ser. LNCS. Springer, 2004, vol. 3046, pp. 574–583.

[6] "Joomla," URL: http://www.joomla.org [accessed: 2014-07-28].

[7] "OpenStack," URL: http://www.openstack.org [accessed: 2014-07-28].

[8] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," NIST, Tech. Rep. SP800-145, Sep. 2011.

[9] NOD32, URL: http://www.eset.com [accessed: 2014-07-28].

[10] ClamAV, URL: http://www.clamav.net/ [accessed: 2014-07-28].

[11] MySQL, URL: http://www.mysql.com/ [accessed: 2014-07-28].

[12] Z. Tan, A. Jamdagni, X. He, P. Nanda, and R. P. Liu, "A System for Denial-of-Service Attack Detection Based on Multivariate Correlation Analysis," IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 2, 2014, pp. 447–456.

[13] Y. Tang, X. Luo, Q. Hui, and R. Chang, "Modeling the Vulnerability of Feedback-Control Based Internet Services to Low-Rate DoS Attacks," IEEE Transactions on Information Forensics and Security, vol. 9, no. 3, 2014, pp. 339–353.

 ISBN: 978-1-61208-367-4

[14] “Three Ways System Downtime Affects Companies and Four Methods to Minimize It,” Globalscape, Tech. Rep., 2014.

[15] “Scan Speeds for 2011/2012 AntiVirus Software,” Antivirus Ware, 2011, URL: http://www.antivirusware.com/testing/scan-speed/ [accessed: 2014-07-28].

[16] “Amazon EC2 Price,” Amazon Web Services, 2013, URL: http://aws.amazon.com/ec2/pricing/ [accessed: 2014-07-28].

[17] F. Machidaa, D. S. Kim, and K. S. Trivedi, “Modeling and analysis of software rejuvenation in a server virtualized system with live VM migration,” Performance Evaluation, vol. 70, 2013, pp. 212–230.

[18] “Infographic: The State of Malware 2013,” McAfee, Inc., Tech. Rep., Apr. 2013, URL: http://www.mcafee.com/us/security-awareness/articles/state-of-malware-2013.aspx [accessed: 2014-07-28].

[19] “3% to 5% of Enterprise Assets Are Compromised by Bot-driven Targeted Attack Malware,” Damballa, Inc., Tech. Rep., Mar. 2008, URL: http://www.prnewswire.com/news-releases/3-to-5-of-enterprise-assets-are-compromised-by-bot-driven-targeted-attack-malware-61634867.html [accessed: 2014-07-28].

[20] V. Chebyshev and R. Unuchek, “Mobile Malware Evolution: 2013,” Kaspersky Lab ZAO, 2013, URL: http://www.securelist.com/en/analysis/204792326/Mobile_Malware_Evolution_2013 [accessed: 2014-07-28].

 ISBN: 978-1-61208-367-4

Design of Mobile Services for Embedded Platforms

Guy Lahlou Djiken
Laboratory of Algorithms, Complexity and Logics, LACL, UPEC University
Créteil, France
guy-lahlou.djiken@lacl.fr

Sanae Mostadi
Ecole Supérieure d'Informatique Appliquée à la Gestion, ESIAG, UPEC University
Créteil, France
mostadis@miage.u-pec.fr

Fabrice Mourlin
Laboratory of Algorithms, Complexity and Logics, LACL, UPEC University
Créteil, France
fabrice.mourlin@u-pec.fr

***Abstract*—The design of distributed applications requires theoretical knowledge and hands-on experience. Our work is about distributed applications based on embedded platforms such as smartphones or tablets. We define a software chain development from design to implementation where services are designed through interface diagrams and component diagrams. From these declarations, we are able to generate software descriptions into two languages. Android Description Language (AIDL) is utilized for local services to an embedded platform. Web Application Description Language (WADL) is utilized for remote services. Such services are called from one platform to another one. The first kind of description allows developers to create Android services. Then, WADL description provides all the features for building Restlet Web services. We applied our strategy to the design and building of a case study on medical picture set management. Embedded tablets can take pictures during the users' activities. Local services allow users to display their medical picture through specific viewers. Remote services are set to expose these data to specific medical material. So, we provided a way to exchange technical data from well spread platforms to medical application servers.**

Keywords-mobility; data collection; mobile service; distributed application.

I. INTRODUCTION

Tanenbaum defines a distributed system as a "*collection of independent computers that appear to the users of the system as a single computer*.". This means that two features are essential: independent and suitable software for hiding the architecture to the users [1].

We consider a distributed system as a collection of autonomous computers linked by a network and using software to produce an integrated computing facility. The size of a distributed system can belong to a local area network (10's of hosts) or a metropolitan area network (100's of hosts) or a wide area network (internet) (1000's or 1,000,000's of hosts). The key characteristics of such distributed systems are the resource sharing where data source or external device are used by applications. Then, the use of open standard allows building applications which need to have the components of a solution work together [2]. The concurrency property is also important; in fact, multiple activities are executed at the same time [3]. This reduces latency and allows hiding blocking with some computing.

The scalability in size deals with large numbers of machines, users, tasks, etc. This property occurs also in a location with geometric distribution and mobility [4]. The subject of our work is the design of distributed applications based on services. When considering scalable application design, a service helps to decouple functionality and think about each part of the application as its own service with a clearly defined interface. For Service Oriented Architecture (SOA) [5], each service has its own distinct functional context, and interaction with anything outside of that context takes place through an abstract interface, typically the public-facing Application Programming Interface (API) of another service.

Building a system on a set of complementary services decouples the operation of those pieces from one another. This abstraction helps establish clear relationships between the services, its underlying environment, and the consumers of that service. Our work is about the use of services which are web services or embedded services. Both types occur into real projects, and it seems to be essential to adopt the same design approach. In Section II, we present our methodology for specifying both types of services. Section III is about the use of intermediate representation between design charts and computer representations. The following section specifies a way to provide an implementation. The two last sections are dedicated to a case study we built on the management of the pictures with their localization. Finally, we sum up about the results we explained in this paper.

II. DESIGN OF DISTRIBUTED SERVICES

Client/server, 3-tier and n-tier distributed applications and cloud computing, open up new opportunities and ways to design systems and develop applications. The design challenge is the main step of the life cycle of any project. The definition of message exchange pattern is essential for the declaration of each remote service. An object-oriented modelling approach is often used to describe business requirements, identify components, their interactions and placement in a multi-tier environment.

We have chosen Unified Modelling Language (UML) [6] [7] as a specification language. There are a lot of charts which can help designers for requirement specification. We have selected deployment diagram for architecture level and how materials are linked. Next, the use of component diagram is the core of our methodology with the specification of interfaces and the declaration of signatures.

 ISBN: 978-1-61208-367-4

A. Design step of distributed services

1) A service approach

Similar to other distributed applications, Web services have a specific structure and behavior. The structure is the static part of Web services, which is composed of the candidate classes and their associations. The behavior is called the dynamic part. It represents how the Web service is executed in terms of sending requests, preparing responses to these requests, and how they will be sent back to the clients.

The UML gains greater acceptance among software designers, not only because of its standardization by the Object Management Group (OMG) [8], but also because of the high support from tool vendors, such as IBM and Oracle.

2) First step in our case study

Along our paper, we use a case study about the management of pictures which are taken with mobile devices such as smartphones and tablets. The main goal for an end user is to know precisely where a given picture is. More precisely, if several devices are used in a lab, it could be convenient to localize the pictures on the devices without any upload of working pictures on a common data server.

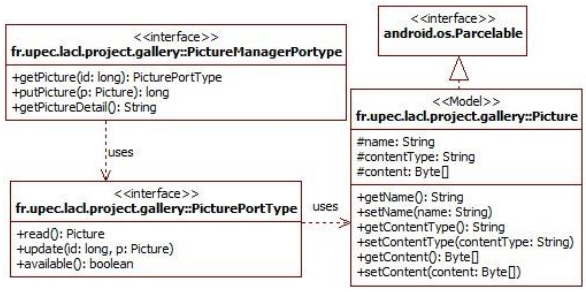


Figure 1. Precise declaration of interface and signature

The main goal of the Web service requirements analysis task is to capture and gather the requirements for the target Web service. This includes the identification of the precise services that have to be provided. This means that UML interfaces are defined in package structure. For instance, assume a context where a set of pictures has to be exposed to a network with HTTP methods. So, Figure 1 describes what will be the first step of the requirement specification.

This short example stresses 2 main tasks: the naming and the signature definition. Type and name of the domain and co-domain are essential to the future implementation and the clients. All these definitions are relative to a namespace (in our example `fr.upec.lacl.project.gallery`). This allows reducing name conflict. A package structure is an ideal entry point into a project dictionary.

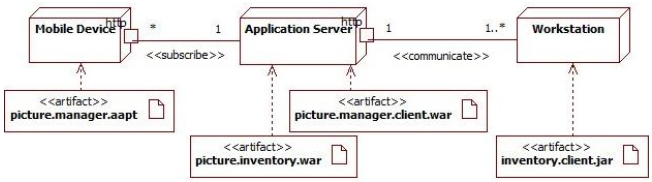


Figure 2. Deployment diagram

On another side, a material description provides all the details useful for the deployment step. In our previous example, the occurrences of the service are deployed on mobile devices. The clients could also be installed on mobile platforms or workstations. In Figure 2, a potential deployment diagram is described as a mobile application server deployed over a mobile device. Its client is installed on an application server. When all data are collected about the pictures, the other artifact, called `picture.inventory.war` deployed over the application server, can answer to the requests of the standard clients.

From this view, we define several artifacts. They play the role of deliverables. Each of them will provide one or more components. A component diagram gives a snapshot of a runtime. Each component has provided interfaces and also dependencies on other parts of the software. Also, we can check how precise the requirements are defined. This allows defining the used network protocol and the message exchange pattern. For instance, the requests to the `PictureManager` service is considered synchronous and parameters are exchanged through an XML format

This component diagram is also the support to express no functional properties such that the maintainability of the set of services and the management of several versions. All the components follow the OSGi specification (Open Service Gateway Interface) [9]. A feature of OSGi technology is its portability since it can be implemented both in the terminal board so that in conventional applications or servers [10]. In this context, the OSGi technology is designed to address the other no functional aspects, such that to enable the management of complex applications and to improve the quality of service applications for administration to warm (see Figure 3).

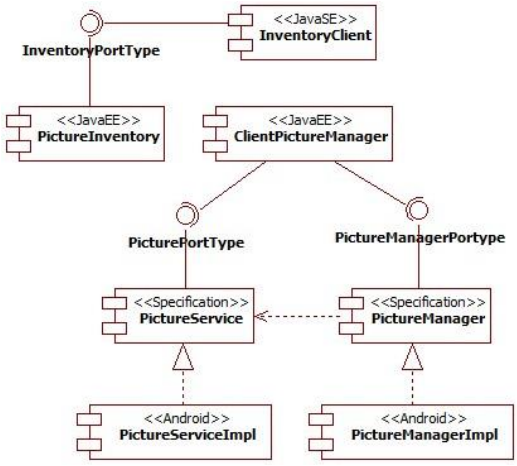


Figure 3. Software architecture of case study

In Figure 3, all components are placed. The naming convention allows readers to understand the correspondence between components and artifacts. There are three kinds of components depending on the kind of deployment node. This diagram highlights the roadmap of our development. So, because the Figure 2 requires different kinds of platform,

then, the next refinements are going to provide more details about the technical features.

B. Integration testing

The integration testing is a level of the software testing process where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. In our context, it means the integration of the three parts: mobile part, server part and a client part. This level of test can be considered as business routes where each of them is a use of our distributed application. In Figure 4, we describe the integration scenario where the application server sends requests to mobile platforms and collects the URLs of pictures and their technical features.

This sequence diagram plays the role of validation after the integration of all the components and their deployment on to the set of materials.

We also use such kind of diagrams when we study the impact of a scenario on the other behaviors of the application server. For instance, the problem can be to understand what the consequences of the data collections are during the subscription of other mobile devices. It seems to be obvious to require that the main business functionalities have to be isolated and the use of one mobile device is independent with the use of another one.

Figure 4 shows the interactions between a tablet and the application server. First, the mobile device is registered and a collector service validates the availability of all the data around the pictures (content, format, identification, localization, etc.). This diagram can be extended with the introduction of other mobile devices or the interaction with other scenarios, but this will introduce some noise into the description and the role of such diagram will be reduced.

III. INTERMEDIATE REPRESENTATION

From the previous set of diagrams, we have to continue towards a more technical representation. As we can observe, this distributed application is based on the use of remote service. These services are clearly defined and depending on the kind of platform, we use a precise approach.

A. AIDL services

The IDL (Interface Definition Language) is generally language independent for the service specification. It is used theoretically for generating C++ or Python stub code from it. The Android one is Java-based though, so the distinction is subtle. One difference is that there is only a single interface in an *.aidl* file, while Java allows multiple classes/interfaces per Java file. There are also some rules for which types are supported; so, it is not exactly the same as a Java interface, and it is not allowed to use one instead of AIDL.

In the context of mobile programming, a service is an application component that runs in the background without a user interface. In our case study, the picture manager can perform data collection by using a background service to prepare data for a foreground application. It means another application of the mobile device. This is quite important because the consequence is that a service built from AIDL cannot be used remotely.

Services work in the background, even though the application is running neither in foreground nor background. A service might handle long running tasks like network connections or retrieving database records with the help of content provider from the background. In our case study, two interfaces are defined to expose services on mobile platform:

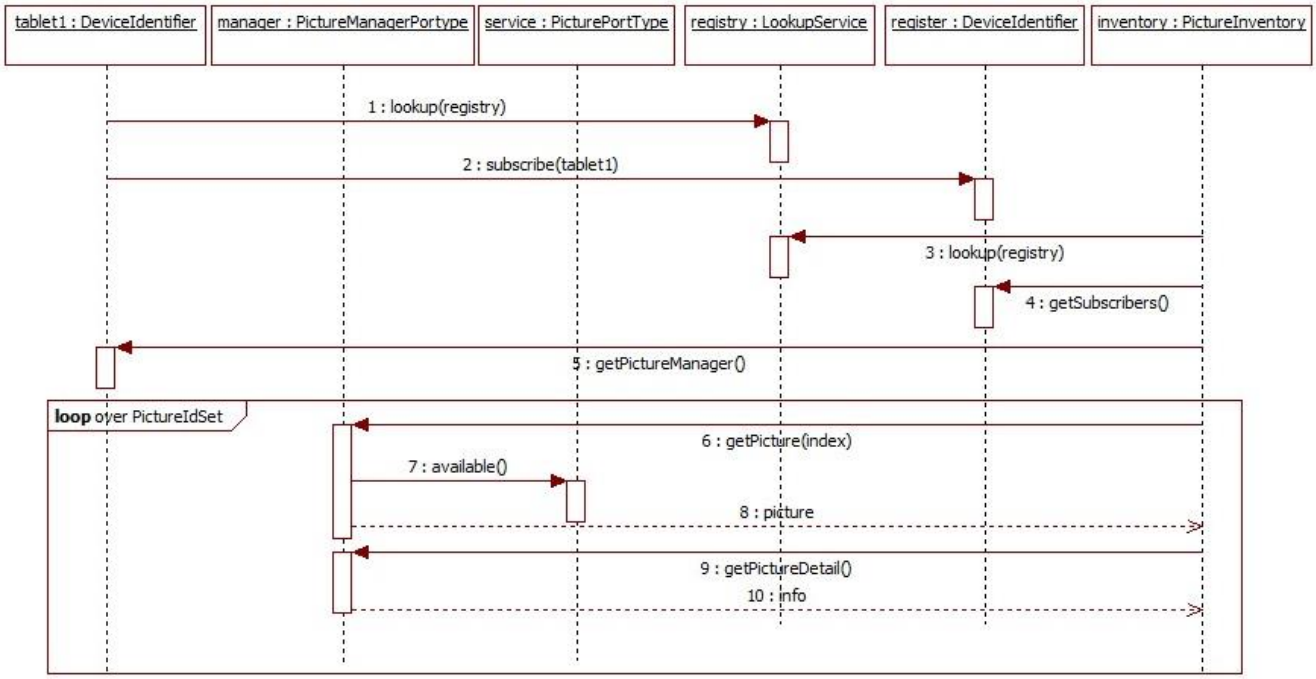


Figure 4. Interaction diagram as integration test

these are `PictureManagerPortType` and `PicturePortType` (see Figure 1). So, from these declarations, we transform them into two .aidl files.

These files (called PictureManagerPortType.aidl and PicturePortType.aidl) define the interfaces that declare the methods and fields available to a client. AIDL is a simple syntax that lets designer declare an interface with one or more methods, that can take parameters and return values. These parameters and return values can be of any type, even other AIDL-generated interfaces. Then, the AIDL compiler creates an interface in the Java programming language from the AIDL interfaces. These interfaces have an inner abstract class named Stub that inherits the interface and implements a few additional methods necessary for the IPC call (Inter Procedure Call).

The next step is to create two classes that extend our previous interfaces `PictureManagerPortType.Stub` and `PicturePortType.Stub` implements the methods we declared in our .aidl file. Then we extend the Service class and override `Service.onBind(Intent)` to return an instance of one of our classes that implements one of our interfaces. The parameter intent plays the role of incoming message. The corresponding AIDL descriptions of Figure 1 are given Figure 5.

The primitive types are in direction by default. We limit the direction to what is truly needed, because marshalling parameters is time expensive. We have a class called `Picture` that we would like to send from a client process to the implementation process through an AIDL interface. We have made the Picture class which implements the Parcelable interface. The consequence is the overriding of the method public void `writeToParcel(Parcel out)` that takes the current state of the `Picture` and writes it to a parcel. The dual method is the method `public void readFromParcel (Parcel in)` that reads the value of a parcel into a `Picture`.

```
package fr.upec.lacl.project.gallery;

interface PictureManangerPortType {
    PicturePortType getPicture(long id);
    long putPicture(in Picture p);
    String getPictureDetail();
  // other methods are added in the case study.
}
```

```
package fr.upec.lacl.project.gallery;

// Declare Picture so AIDL can find it, knows
// that it implementsthe parcelable protocol.
parcelable Picture;
```

```
package fr.upec.lacl.project.gallery;

interface PicturePortType {
        Picture read();
        boolean update(long id, in Picture p);
        boolean available();
  // other methods are added in the case study.
}
```

Figure 5. AIDL output files

B. REST services

The use of AIDL is required because of application sandboxing. Each application in Android runs in its own process. An application cannot directly access another application's memory space. In order to allow cross-application communication, Android provides the inter-process communication protocol. IPC protocols tend to get complicated because of all the marshaling/unmarshaling of data that is necessary, but it has also a main limit: it is not possible to use it in a remote manner.

Today, a remote access is a common requirement, but the installation of a web server on a mobile platform is not so natural. Also, we propose to use remote access by the use of the REST service through the use of Google implementation called Restlet. REST stands for Representational State Transfer. It relies on a stateless, client-server, with cache communications protocol, and in generally all cases, the HTTP protocol is used. REST is an architecture style for designing networked applications. The idea is that, rather than using complex mechanisms such as CORBA [11], RPC [12], or SOAP [13] to connect between machines, simple HTTP is used to make calls between machines.

As a programming approach, REST is a lightweight alternative to Web Services and RPC (Remote Procedure Calls) and Web Services (SOAP, WSDL [14], and others). Much like Web Services, a REST service [15] is platform-independent, language-independent, standards-based runs on top of HTTP, and can easily be used in the presence of firewalls.

There are several reasons for having a Web server on a mobile phone. The main one is to allow third-party applications, on other phones or other platforms to access to the phone remotely. This requires strong security mechanisms that are provided in part by the Restlet framework as well as network level authorizations by the carrier. We have decided to apply a Proxy design pattern to hide Restlet mechanism. So, each AIDL service is equipped with a Restlet service. To sum up, the AIDL implementation is used as a local facet on the mobile device and the Restlet implementation can be considered as a remote facet from other platforms.

In accordance with the Proxy design pattern, we have declared a subclass of the `ServerResource` class which belongs to the Restlet framework. Our class is called `PicturePortTypeResource` and has an attribute which is the previous AIDL implementation. Both classes implement the same business interface, but this last one provides our local service on the http protocol as a web resource. Figure 6 shows the main changes. Two technical packages are drawn to precisely the role of our technical classes.

Now, this mobile part is accessible from other mobile devices and also from workstation and application server if necessary.

 ISBN: 978-1-61208-367-4

IV. CODE CONSTRUCTION

We design the embedded part in the respect of such properties, such that the independence of the layers and interoperability. It means that the client part of the previous service does not know any technical details of our solution. This preserves the client from the changes of the new versions.

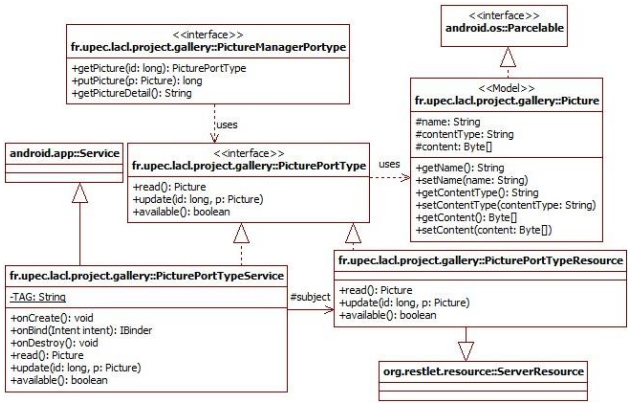


Figure 6. Design class diagram of the mobile part

A. JavaEE implementation

As explained previously, the middle layer is the pilot of the data collection. After the subscription of a mobile device, requests are sent periodically from the application server to the mobile device. Applications that model business work flows often rely on timed notifications. We schedule a timed notification to occur at time intervals. Then, the collected data are stored on the application server. Of course, other mobile devices can subscribe to that picture manager service even if several data collections are running. Both functionalities are isolated.

Another artifact is deployed on this application server: it is the inventory service. It is a stateless component which answers to the presentation layer running on a client workstation. The role of the inventory service is to answer to the client about the previous data collections. For instance, assume several mobile devices are previously registered, so a client can ask precisely to know where a picture, called "picture1", under a JPEG format is. The structure of that part is more convenient: it is a three tier layer. These different responsibilities of an application are broken up into distinct tiers, typically:

- The integration tier for data transformation and persistence services. The persistence unit is about the details which are collected during the data collection.
- The business tier for the validation, business rules, workflow and interfaces to external systems. The request is expressed by a subset of the features of the pictures. This means the content type, the size the annotations, etc.
- The presentation tier for user interface generation and lightweight validation. The web panels allow the requester to define his need.

The requests between the presentation and business layers are synchronous over TCP protocol, but a message broker is used to separate client and service. The exchanges are totally asynchronous between the business and the integration layers. This is essential because the integration part can be considered as a cache of the database for several web applications.

B. JavaSE implementation

First, we use a web explorer to send http request and to display html tier. This display is a default graphical user interface used to send requests about the location of images. Next we have provided an API to develop new requests into programmatic clients. This is particularly useful for the automatic functional tests. This allows us to replace the use of Selenium tool of our own test application.

Our API allows also other developers to program new client tiers. It is based on the use of REST services which send requests to our business tier. Because, we have chosen a REST implementation with the WADL generation (Web Application Description Language), other developers can build their own version of our API. Also, SOAPUI tool [16] provides an easy way to create test suites of our business tier.

Our next case study is built with a lightweight client tier. In this context, the user is sure that the web client is well suitable for the version of the business tier. Moreover, a comparison with other testing tool can be done especially for performance measures.

V. CASE STUDY

As we explained in our contribution, our case study is about the management of the pictures on Smartphone. Several embedded devices are used, for instance, in a lab or in a classroom. So, a distributed tool is necessary to locate precisely where the pictures are. More generally, such kind of tools is useful for the whole management of the pictures. This means collect, remove, transfer, duplicate or transform to an appropriate format.

A. Deployment view

Before starting our case study, we have to deploy all artifacts on given computer as mentioned Figure 2. Next, services have to be started by local servers. So, observations and measures could be done by a tester.

1) Mobile data tier

Under Android 4.2 operating system, the mobile devices are used by members of a laboratory to take photos. The camera records the pictures into a gallery where each of them corresponds to a separate file with a set of features (name, format, size, date, owner, etc.). Because, a gallery can be considered as a set of pictures, each picture has an own name for their identification. Often, the name is generated by the software component which manages the camera. This means that the name is not easily known by the scientist.

For a test phase, the first activity is to take several photos. And then, register the mobile device as a data tier to a business server. This will engage a set of REST services as and points to the gallery of photos

 ISBN: 978-1-61208-367-4

2) Business tier

Its first objective is to ready for receiving registration of all the mobile devices. From its point of view, the mobile devices are considered as a distributed data set of pictures. Concurrently, it performs a data collection about the features of the photos. This is not a collect of the photos because this will spend too much time. But, this activity is to bind all the features such as localization into a registry for future requests. The inventory activity is managed by a timer. Also, regularly, a mobile device receives requests about new pictures if there are until the end of its registration onto the business server.

A third activity is to answer to the end users who want to localize the photos which are taken during a given period of time. Additional conditions can be set such as the content type, the dimension of the picture, the size of the file, etc.

3) Client tier

In the test phase, we use a web client for sending the requests. This client is received by sending an HTTP request from a navigator. It allows end users to define precisely the photos that they want to have access. The answer of a request is a set of links. They can be used to access to the embedded devices and the concrete photo. So, by the end of a test, this means: a request and a click on a hypertext link, a photo is displayed in the web browser of the end user.

B. Artifact deployment

1) Mobile data tier

In order to install third party applications to our Android phone, we need to install APK (Android Package, files). The way we usually do is like the next iteration, but ir is for testing:

- Plug in an USB cable to a PC and mount a SD card on the computer
- Get the APK file somewhere on the SD card on the phone
- Unmount the SD card on the PC, allowing the phone to see the SD card contents again
- Use Astro File Manager or some similar app to browse to that file on the SD card and select it, which will prompt us if we want to install the app on the phone.

For the end users, we have defined a more simple strategy based on the use of the local repository. We deploy the .apk file on a local server (apache http server) with a static IP to make the file available for download. Now the end user has to open the download link of the apk file in his mobile browser. The device will automatically start the installation after the download completes.

2) Business tier

We use an application server called JBoss where our applications are installed though an ear files (Enterprise Application aRchive). The standard configuration of JBoss provides a system for deploying applications very simple and convenient, but not necessarily suitable for a production environment.

As standard, the deploy directory is a configuration where deploying services, components and applications. Just include a file according to the specific type of component specifications for JBoss deployment take into account. It is possible to deploy the files to deploy directory or its subdirectories. Each file type is taken into account by an appropriate service deployment. The EARDeployer service is used for our two main components: the registration of tablets and the data collector.

The AbstractWebDeployer service is used for the Web application called by the client. It is implemented for the servlet container TomcatDeployer. The archive files are in the format war (Web ARchive).

3) Client tier

In the test phase, we use a web client for sending the requests. This is a set of JSP pages which belongs to the previous Web application. Also, the client tier is just a Web browser which is already installed on the computer of the client.

We also use Java Web Start which is a mechanism for program delivery through a standard web server. The Java GUI client is downloaded to the client and executed outside the web browser. The GUI client does not need to be downloaded again on the next run. If the GUI client is updated, a new version will be downloaded automatically. The jar file contains an XML descriptor with an XML schema. It specifies the resources needed to run Java Web Start applications. It defines also the URL location of the jar file, VM arguments and other resources that JRE on the client side should know to start Java Web Start GUI client.

Such GUI client that needs access to system resources, like file system, network connections, etc., need to be signed. Also, we generate a keystore (certificate) and attach it to the jar file. After that, an end user is able send request to the business tier and also to access to a mobile device.

C. Measures

Measuring the execution time is a really interesting, but also complicated topic. To do it right in Java, we have to know a little bit about how the JVM works: generation decomposition and so on. But, we do not have the same VM on all the nodes of the network. The mobile devices have a DVM (Dalvik VM), the business tier and the client tier have a JVM (Java VM); but the versions are not correlated.

Also, we use a "ready to run" benchmarking framework that addresses many of our issues [17].

1) Measuring method execution time: The framework's essential class is named Benchmark. It is the only class that we use for the computation of measures; everything else is ancillary. Client and business tiers are observed by instances of the Benchmark class. We supply the code to be benchmarked to the Benchmark constructor. The benchmarking process is then fully automatic. Then, we generate a result report. The only restriction is that the code be contained inside a Callable or Runnable. Otherwise, the target code can be anything expressible in the Java language.

2) Business tier Measures: There are two sets of measures. One is about the requests between the mobile devices and the application server. There two main tasks are: one is the registration of the mobile devices, the second

 ISBN: 978-1-61208-367-4

is the data collection which is started and ended by the application server.

The other set is about the treatment of the requests of the clients. Each request is received and treated by a business action which is also a `Runnable` instance. This means that we have measures on it. Both are interesting and their observations involve future improvements.

3) Results

Table 1 presents measures of `RegistrationTask` class. It is a `Callable` subclass and its method is invoked when a mobile device needs to belong to the community of the mobile data tier. Next, a data collection will be occurred.

TABLE I. REGISTRATION OF MOBILE DEVICES

Measures	Method execution time		
	First time	*Mean time*	*Standard deviation*
Registration	112.901 ms	108.501 ms	725.510 µs

In the meantime, we have additional information on it: deltas: -35.205 µs,+46.206 µs).

For the standard deviation execution time, we have the info: deltas: -161.405 µs, +361.108 µs

Table 2 presents measures of `DataCollectionTask` class. It is a `Runnable` subclass and its behavior is managed by a timer. Each interval of time a data collection is started on a given mobile device. By the end, the changes are updated on the business server. This task is not linked to the previous one and several data collections are started concurrently in a manner that there is no effect from one data collection onto the other ones.

TABLE II. DATA COLLECTION ON A MOBILE DEVICE

Measures	Method execution time		
	First time	*Mean time*	*Standard deviation*
Data collection	225.910 ms	220.050 ms	555.004 µs

In the meantime, we have additional information on it: deltas: -31.520 µs,+41.602 µs).

For the standard deviation execution time, we have the info: deltas: -124.040 µs, +302.088 µs

Table 3 presents the measures of the `ClientRequest` class. It is also a `Runnable` subclass and its method is invoked when the end user sends a request about the url addresses of several photos. Next, all the features of the user request are parsed and a result is computed from the previous data collections. Then, an answer is built with a set of URL instances. Each URL instance is a REST call to a service deployed on a mobile device.

TABLE III. CLIENT REQUEST ABOUT PHOTO ON DISTRIBUTED DEVICES

Measures	Method execution time		
	First time	*Mean time*	*Standard deviation*
Client request	164.621 ms	158.921 ms	605.233 µs

In the meantime, we have additional information on it: deltas: -41.115 µs,+51.261 µs).

For the standard deviation execution time, we have the info: deltas: -103.523 µs, +112.561 µs

VI. ANALYSIS

The first time that `RegistrationTask` instance was called, it took 112.901 milliseconds to execute. A point estimate for the mean of the execution time is 108.501 milliseconds. The 95% confidence interval for the mean is about -35/+46 microseconds, which is relatively narrow, so the mean is known with confidence.

A point estimate for the standard deviation of the execution time is 725.510 microseconds. The 95% confidence interval for the standard deviation is about -161/+361 microseconds about the point estimate, namely [235.389, 1086.51] µs, which is relatively wide, so the deviation is known with much less confidence. In fact, the warning at the end says that the standard deviation was not accurately measured. The result also warns about the outliers. They are no significant in this case because the scenarios contain network connections. This involves blockings and time consuming only for negotiation between mobile devices and business server.

In the case of the data collection, the first time that `DataCollectionTask` instance was called, it took 225.910 milliseconds to execute. A point estimate for the mean of the execution time is 220.050 microseconds. The 95% confidence interval for the mean is approximately -31/+42 microseconds, which is relatively narrow too, so the mean is known with confidence.

We guess the standard deviation of the execution time is 555.004 microseconds. The 95% confidence interval for the standard deviation is about -124/+302 microseconds about the point estimate, namely [430.964, 857.092] µs, which is less wide than the previous case. So deviation is known with much confidence. In fact, the warning at the end notes that the standard deviation comes from the size of data which is collected. The result also warns about the variability in the measurement. The latter is sometimes excluded from the data set.

The last case is about request treatment. The first time that `ClientRequest` instance was called, it took 164.621 milliseconds to execute. A point estimate for the mean of the execution time is 158.921 microseconds. The 95% confidence interval for the mean is approximately -41/+51 microseconds, which is relatively narrow too, so the mean is known with confidence.

Then, we guess the standard deviation of the execution time is 605.233 microseconds. The 95% confidence interval for the standard deviation is about -103/+112 microseconds about the point estimate, namely [501.71, 717.794] µs, which is relatively few. So, it is known with confidence. In fact, the warning at the end notes that the standard deviation comes from the number of requests which are received by the Web application. The result also indicates an experimental error because of the latency of the network. When we compute other measures on a sample with a bigger volume of

 ISBN: 978-1-61208-367-4

requests, then this overhead time is hidden or recovered by the computation of the answers.

VII. CONCLUSION

We have presented in this document our approach to the design (D), the implementation (I) and the evaluation (E) of mobile applications based on services. It was shown that there are two families of services: some of them are local and the others are called from outside the mobile platform. Our Design is based on the use of UML diagrams and stereotypes to identify interfaces and the locality.

The Implementation is based on Java programming and the use of frameworks such as Restlet and Android. We have shown how to refine he diagrams towards a more technical description. A designer can sketch his applications with the use of local or remote services.

The Evaluation is also described by interaction diagrams which will become a test suite. We have built a case study based on our approach. It highlights all kinds of services (local and remote). So, interoperability is insured by the use of XML messages.

To sum up, our approach, called D.I.E. validates our design choice. Our experiments highlight the use of mobile devices as mobile data tier. As the number of embedded devices increases, our prototype shows that our software protocol supplies a way to exploit data on mobile devices without big data transfers.

REFERENCES

[1] J. N. Herder, H. Bos, B. Gras, Ph. Homburg, and A.S. Tanenbaum: Reorganizing UNIX for Reliability. "Asia-Pacific Computer Systems Architecture Conference", 2006, pp. 81-94

[2] M. Gould, M.A. Bernabé, C. Granell, P.R. Muro-Medrano, and J. Nogueras, Reverse engineering SDI: Standards based Components for Prototyping, "8th EC-GI & GIS Workshop ESDI" - A Work in Progress Dublin, Ireland July. 2002. pp. 3-5

[3] S.U. Khan, A.Y. Zomaya, and L.Wang, "Scalable Computing and Communications": Theory and Practice, Wiley-IEEE Computer Society Press. January 2013,

[4] T. Erl. Service-oriented architecture: concepts, technology, and design. Pearson Education India, 2005.

[5] J. Rumbaugh, J. Ivar, and B. Grady. Unified Modeling Language Reference Manual, The. Pearson Higher Education, 2004.

[6] M. Randles, D. Lamb, A. Taleb-Bendiab : A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing, : Proceedings of the 2010 IEEE "24th International Conference on Advanced Information Networking and Applications Workshops", WAINA '10, IEEE Computer Society, Washington, DC, USA, pp. 551-556.

[7] S. Th□one, R. Depke, and G. Engels. Process-Oriented, Flexible Composition of Web Services with UML: " Advanced Conceptual Modeling Techniques". pp. 390-401. SpringerLink, October 2003.

[8] Models, Object. "Object Management Group." Draft 0.3, January 12 (1995).

[9] Alliance, OSGi. Osgi service platform, release 3. IOS Press, Inc., 2003.

[10] P. Kriens, "OSGi Service Platform, Enterprise Specification", Version 4.2, aQute publisher, ISBN 978-90-79350-06-3.

[11] J. Siegel. CORBA 3 fundamentals and programming. Vol. 2. Chichester: John Wiley & Sons, 2000.

[12] N. B. Jay. Remote procedure call. No. CSL-81-9. Carnegie-Mellon Univ. Dept. Comput. Sci., 1981.

[13] M. Gunnar, et al. "Simple object access protocol." U.S. Patent No. 6,457,066. 24 Sep. 2002.

[14] C. Erik, F. Curbera, G. Meredith, S. Weerawarana. "Web services description language (WSDL). 2001. pp. 1-1.

[15] K. Rohit, and R. N. Taylor. "Extending the representational state transfer (rest) architectural style for decentralized systems." Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on. IEEE, 2004.

[16] K. Charitha. Web services testing with soapUI. Packt Publishing Ltd, 2012.

[17] B. Boyer, "Robust Java benchmarking: Introducing a ready-to-run software benchmarking framework", Programmer Elliptic Group, Inc, IBM Red book

 ISBN: 978-1-61208-367-4

The Quantification of Risk Factors for Predicting Diabetic Cystoid Macular Edema based on a Hierarchical Approach

Eun Byeol Jo*, Ju Hwan Lee*, Jong Seob Jeong*, Byeong Cheol Choi† and Sung Min Kim*
*Department of Medical Biotechnology, Dongguk University-Seoul
Seoul, South Korea
{eunbyeol27, ykjhlee, jjsspace}@gmail.com, smkim@dongguk.edu
†Department of Biomedical Engineering, Choonhae College of Health Sciences
Ulsan, South Korea
bcchoi@ch.ac.kr

***Abstract*—This study suggests a novel risk factor extraction method for retina layers based on a hierarchical approach to distinguish Diabetic Cystoid Macular Edema (DCME) from optical coherence tomography scans. For this, a total of 80 subjects composed of 30 normal and 50 DCME patients were selected. To estimate evaluation variables, a hierarchical approach-based feature extraction algorithm was employed. Evaluation variables were classified into the Total Retina (TR), the Inner Retina (IR), the Photoreceptor Outer Segment (POS), the Outer Retina (OR), the Ganglion Cell (GC), and the Retinal Nerve Fiber Layer (RNFL). The experimental results show the reliable performance of the proposed approach in discriminating DCME from normal subjects. The proposed method could differentiate changes in the thickness of the IR and the POS between the normal and DCME groups. In addition, the most significant degeneration was observed in the central macular area. These results suggest the clinical applicability of the proposed method to the diagnosis of DCME.**

Keywords-diabetic cystoid macular edema; optical coherence tomography; retina layer; thickness; hierarchical approach

I. INTRODUCTION

Diabetic retinopathy is one of the most frequent complications caused by diabetes. The prevalence of diabetes has increased in an aging society, and patients with diabetic retinopathy have also been increasing. Among various types of diabetic retinopathy, Diabetic Cystoid Macular Edema (DCME) is a major cause of vision loss [1]. DCME increases the thickness of the retina by accumulating liquid inside through the collapse of the retinal barrier and causes macular degeneration at the center of the retina where images are focused on [2].

The diagnosis of DCME is usually performed using the Retinal Thickness Analyzer (RTA) [3], Heidelberg Retina Tomography (HRT) [4], and Optical Coherence Tomography (OCT) [5]. Among these techniques, OCT is known as the gold standard for diagnosing DCME [6] because of its superior sensitivity. OCT also allows for the quantitative measurement of retina lesions and structures. However, OCT cannot identify various intra-retinal structures accurately, and the accuracy of extracting macular thickness is relatively low. In particular, it represents the lowest accuracy for retina nerve fiber layers, where photoreceptors exist [7].

To address these problems, a number of studies have extracted macular thickness based on retinal layer boundaries from OCT images. Mujat et al. [8] proposed a novel boundary extraction approach for smoothing boundaries and reducing the image-processing speed based on the deformable spline algorithm. However, this method cannot process large amounts of data simultaneously. Koozekanani et al. [9] tried to minimize detection errors for retinal boundaries by using the standard Markov boundary model [10]. Bartsch et al. [11] simply extracted the longest boundary appearing continuously in the retina layer based on an improved Markov boundary model. Also, Yazdanpanah et al. [12] successfully segmented the retina and the choroid from OCT images by using a dual-thresholding technique and found the availability of only limited information. Gonzalez et al. [13] tried to identify retina layers by using the Hough transform, but this technique cannot successively extract various thickness values. Chiu et al. [14] extracted retinal layers by taking a graph-search segmentation approach using dynamic programming. Similarly, Yang et al. [15] minimized thickness measurement errors by applying the weight of graph-based dual-scale gradient information. The graph-search segmentation method used in the above two studies has a disadvantage in that it takes a long time to calculate because the number of operations varies widely according to the resolution of images.

This study estimates the risk factors of retina layers using a hierarchical approach-based feature extraction method to appropriately distinguish DCME from the original OCT scan. The rest of this paper is organized as follows: Section 2 develops the algorithm and discusses the background of the experiment. Section 3 presents the results, and Section 4 discusses them. Finally, Section 5 concludes.

II. MATERIAL AND METHODS

A. Image Acquisition

The experiment included a total of 80 subjects composed of 30 normal and 50 DCME patients. These subjects were classified into 50 normal and 100 DCME eyes based on a clinical evaluation with a trained operator. Retina images

 ISBN: 978-1-61208-367-4

were acquired using OCT (Cirrus HD-OCT, Carl Zeiss Meditec, Inc. Dublin, CA). The Institutional Review Board of Dongguk University Hospital approved this study, and all participants gave their consent to participate in the study.

B. Algorithm Development

To extract the thickness of retina layer, a modified hierarchical approach of Koprowski [16] was employed. First, denoised images were obtained by applying median filtering to the original OCT scan (Figure 1(b)). Then, the difference between the original image and the blurred image (image mask) was obtained, and synthesis outcomes for output and original images were printed. Images from the aforementioned step showed a low resolution. To measure the retina layer, the image was decomposed into pixels of *N x N* size *(N=15, 16, 17)* (Figure 1(c)). In addition, thresholding techniques using the Otsu algorithm were applied to separate the threshold in the degraded image (Figure 1(d)). The pixel value of the output image indicated the average value of the decomposed image into this *N x N* size. Based on the average value, the maximum pixel value in each column was derived as follows:

$$L_{DM}(m,n) = \begin{Bmatrix} 1 & if & L_D(m,n) = \max_m(L_D(m,n)) \\ 0 & other & \end{Bmatrix} \quad (1)$$

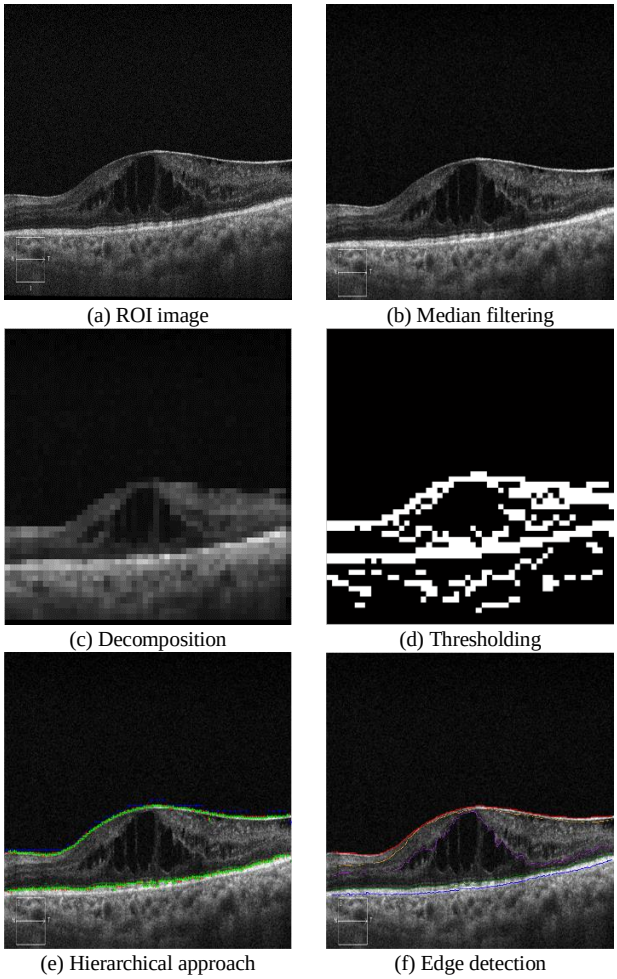
where *m* and *n* refer to rows and columns, respectively. Based on images acquired from (1), the top image boundary was obtained. The lowest boundary of the image was measured using the following (2):

$$L_{DM}(m,n) = \begin{Bmatrix} 1 & if & |L_D(m,n) - L_D(m+1,n)| > P_r \\ 0 & other & \end{Bmatrix} \quad (2)$$

where P_r indicates the threshold in the (0, 0.2) range according to each pixel and satisfies $m \in (1, M-1)$, $n \in (1, N)$. Through this process, the bottom boundary of the retina layer was extracted, and upper and lower boundaries of the retina were obtained $(L_{DM}, L_{DB} = 1)$. The boundary coordinate showing a value of 1 in the output image was defined as $LDB(x) \leq LDM(x)$. The boundary coordinate was the output a total of three different boundary coordinate values based on pixel size. Figure 1(e) shows the synthesis results for the original image and three boundary outputs. This procedure was repeated to minimize errors during the detection of boundaries. The mean value of three boundary values was designated as the final boundary value (Figure 1(f)). All image processing was performed using the Matlab software (R2011b, MathWorks Inc., Natick, MA, USA).

C. Evaluation Variables

The proposed approach was employed to extract six risk factors, including the Total Retina (TR), the Inner Retina (IR), the Photoreceptor Outer Segment (POS), the Outer Retina (OR), the Ganglion Cell (GC), and the Retinal Nerve Fiber Layer (RNFL). The TR was the retina layer from the top boundary to the bottom edge (Figure 2), and the IR was the retina layer obtained by subtracting the OR and the POS



(a) ROI image (b) Median filtering (c) Decomposition (d) Thresholding (e) Hierarchical approach (f) Edge detection

Figure 1. Image processing procedures for obtaining the thickness of retina layer.

from the TR. The POS was the lowest retina layer, and the OR was the thickness of swelling in the retina layer. Cells were distributed in the GCL, which delivered visual information to the brain and was included in the TR. The

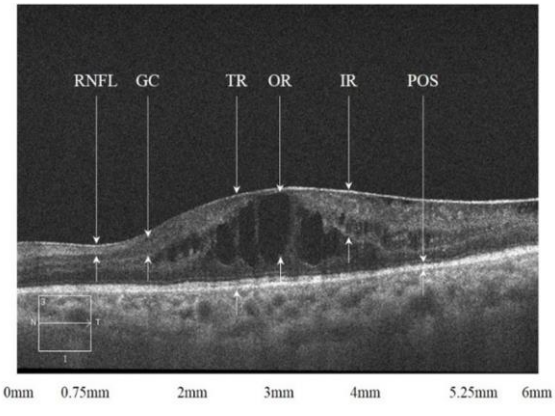


Figure 2. The specific positions of each evaluation variables in the OCT image.

 ISBN: 978-1-61208-367-4

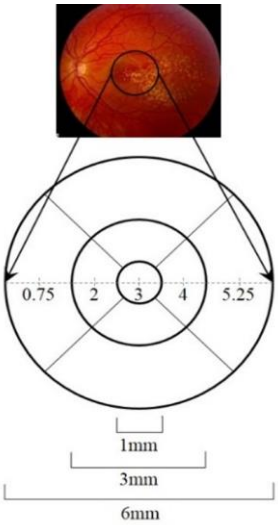


Figure 3. Regions for measuring risk factors from the center of the macula (0.75 mm, 2.00 mm, 3.00 mm, 4.00 mm, and 5.25 mm).

RNFL was the retina layer at the top and included optic nerve cells. Figure 2 shows the position of each evaluation variable in the OCT image.

Each evaluation variable was extracted from the original OCT image captured within a 3 mm radius from the center of the macula. OCT images were divided into a total of five measurements (0.75 mm, 2.00 mm, 3.00 mm, 4.00 mm, and 5.25 mm), and the average value of each measurement was specified as an evaluation variable (Figure 3).

D. Statistical Analysis

Data were analyzed using an independent t-test and a one-way ANOVA based on SPSS (Ver. 12.0 for Windows, Chicago, IL, USA). A p-value less than 0.05 was considered significant.

III. EXPERIMENTAL RESULTS

A. A Comparison of Thickness Extraction Performance between Cirrus HD-OCT and the Proposed Method

To evaluate the performance of the proposed extraction approach, evaluation variables were measured using Cirrus HD-OCT equipment, and statistical significance was compared. The OCT equipment was only able to measure the thicknesses of the TR, the GC, and the RNFL.

In the normal group, the proposed method showed higher reliability than the existing method. The R^2 value, which was used to evaluate the significance of variables, was close to 1 (Figure 4). On the other hand, the R^2 value of most evaluation parameters was close to 1 in the DCME group, but the RNFL showed a low value of 0.0446 (Figure 5(d)). TR and GC values extracted by Cirrus HD-OCT showed significant differences between the normal and DCME groups ($p < 0.05$). However, the RNFL showed a low level of significance ($p > 0.05$).

B. A Comparison of Retinal Layers between Normal and DCME Groups

According to correlations between evaluation variables for the normal and DCME groups, the TR, the OR, the GC, and the RNFL showed high levels of significance in all

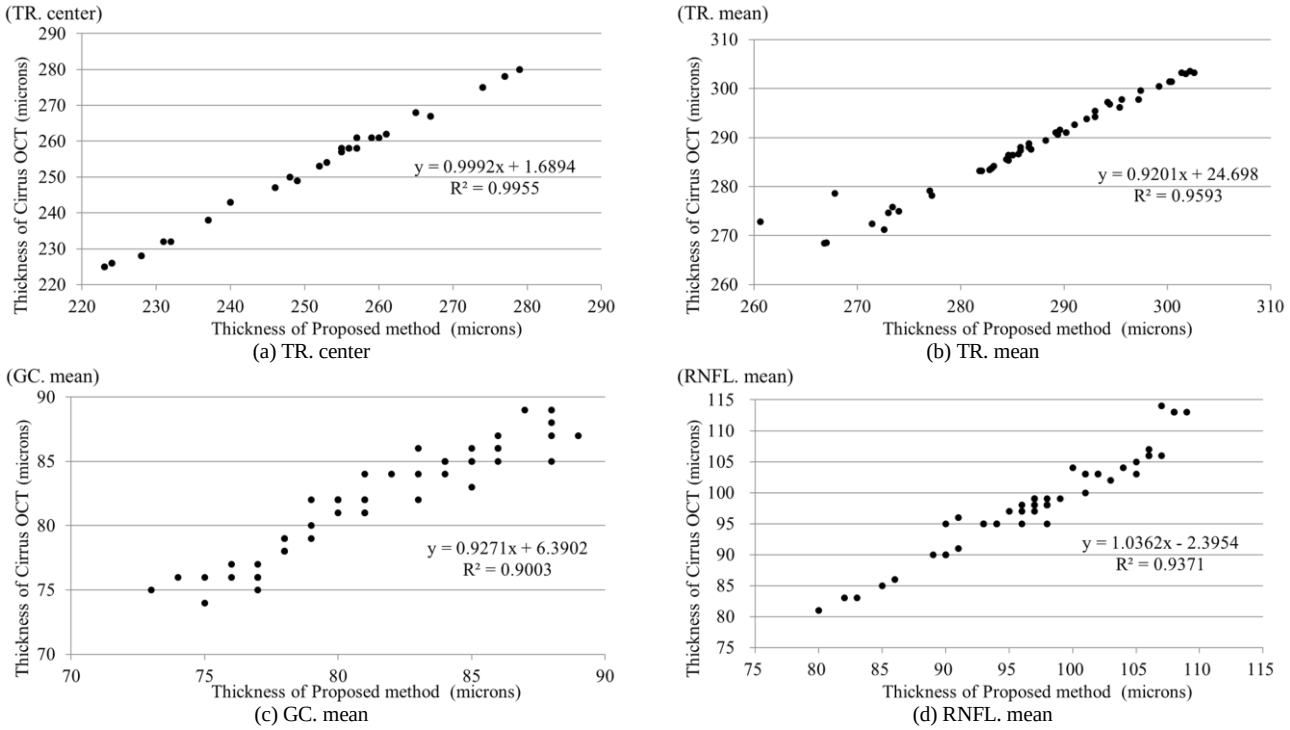


Figure 4. Comparison of the evaluation variables obtained from the Cirrus HD OCT and proposed method for normal subjects

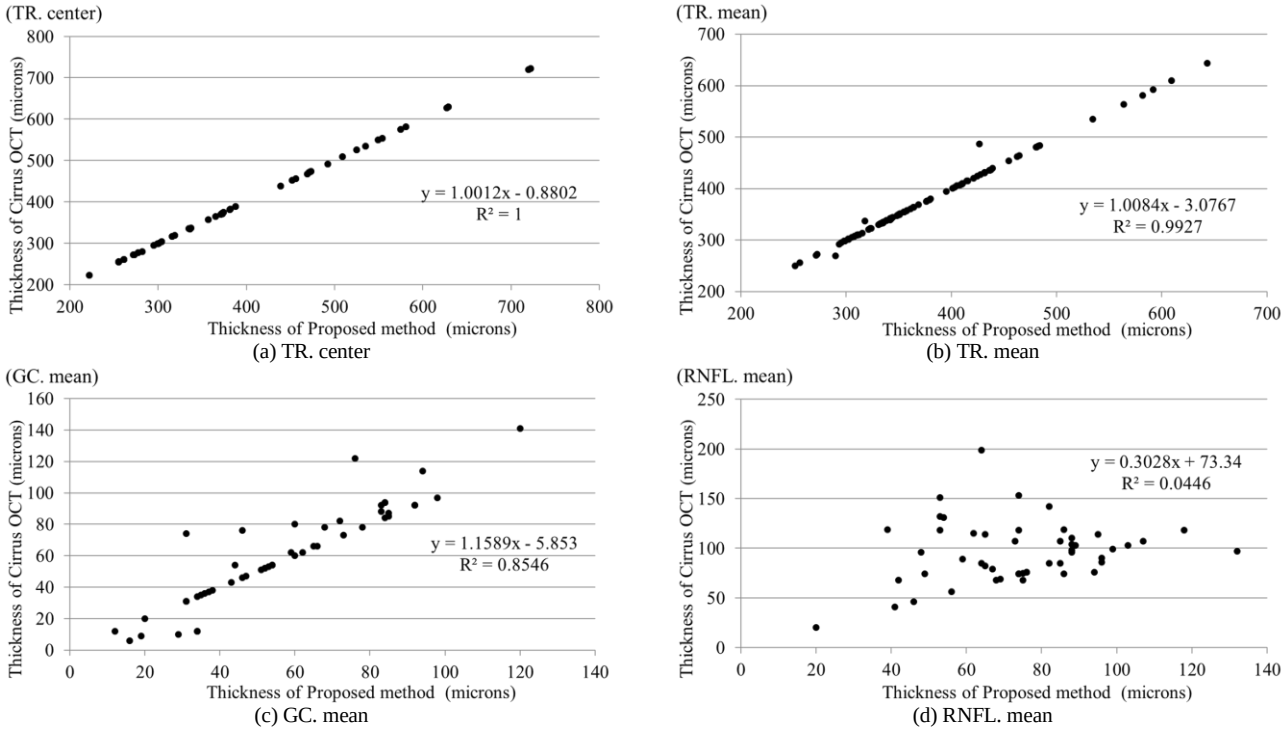


Figure 5. Comparison of the evaluation variables obtained from the Cirrus HD OCT and proposed method for DCME group

macular regions ($p < 0.05$). On the other hand, the IR and the POS showed high levels of significance in the normal and DCME groups only in the central macular region ($p < 0.05$), and the other areas showed a low level of significance.

The TR was thicker in the normal group than the DCME group in all macular regions. In particular, differences increased toward the center of the macula, and the largest difference (148.051 mm) was found at the center of the macular area. The IR was larger in the normal group than in the DCME group (except for the 4.5 mm - 6.0 mm range). The largest difference (78.494 mm) was found at the center of the macula. In addition, the POS was thicker in the normal group, and the difference at the center was 29.421 mm. Finally, the GC and the RNFL were larger in the normal group. The OR could be measured only in the DCME group. For all the aforementioned evaluation parameters, the standard deviation was higher in the DCME group than in the normal group. Table 1 compares evaluation parameters extracted using the proposed method between the normal and DCME groups.

IV. DISCUSSION

The proposed method could measure the thickness of the IR, the POS, and the OR, which could not be obtained using the existing method. It could also measure the TR, the GC, and the RNFL in OCT images. According to a comparison of experimental results based on Cirrus HD-OCT, extracted evaluation parameters were very similar to those based on the existing method. In addition, RNFL values obtained by Cirrus HD showed a low level of significance in discriminating between the normal and DCME groups, unlike in the case of other retina layers, providing no support for the suitability of clinical RNFL data for diagnosing DCME.

According to the statistical analysis of evaluation variables based on the proposed method between the normal and DCME groups, the TR, the GC, and the RNFL showed high levels of significance for the whole area of the macula. This indicates that extracted risk factors were significant predictors of DCME. On the other hand, the IR and the POS showed low levels of significance in some sections, and the center of the macula (clinically the most important area) showed a high level of significance in both normal and DCME groups. This implies the usefulness of the IR and the POS for diagnosing DCME.

According to a comparison of differences in evaluation variables between the normal and DCME groups, the TR was larger in the DCME group. This was mainly because the whole layer swelled up from retinal edema. The IR was much smaller in the DCME group than in the normal group. The decrease in the IR typically implies an increased risk of vision loss, since the IR includes the GC and the RNFL [17]. The POS tended to become thinner with retinal neovascularization, indicating greater damage. Given this, the POS was minutely thinner in the DCME group than in the normal group, indicating the worsening of retinal damage. In addition, the GC and the RNFL were larger in the normal group, indicating an increase in the risk of macular degeneration [18]. Further, the difference between the normal and DCME groups increased significantly in the macular area, the center of the retina, because abnormalities generally occur first in the macular area, where the optic

 ISBN: 978-1-61208-367-4

TABLE I. COMPARISON OF EVALUATION PARAMETERS EXTRACTED FROM OCT SCANS USING THE PROPOSED METHOD BETWEEN THE NORMAL AND DCME GROUPS

	Horizontal distance	Normal group	DCME group	P-value
TR	0~1.5mm	279.625±22.445	343.100±72.930	0.000
	1.5~2.5mm	318.333±13.760	398.520±94.999	0.000
	2.5~3.5mm	252.229±15.383	400.280±126.402	0.000
	3.5~4.5mm	314.917±17.140	394.470±92.306	0.000
	4.5~6mm	268.979±18.589	336.020±64.825	0.000
	Mean	286.817±11.245	374.478±76.799	0.000
IR	0~1.5mm	215.576±24.832	214.846±55.775	0.952
	1.5~2.5mm	251.106±34.584	215.614±63.400	0.004
	2.5~3.5mm	168.793±15.719	90.299±70.487	0.000
	3.5~4.5mm	252.800±14.517	227.090±51.097	0.003
	4.5~6mm	212.302±22.481	221.624±60.098	0.362
	Mean	220.117±12.231	193.893±34.351	0.000
POS	0~1.5mm	63.757±6.796	62.585±19.025	0.714
	1.5~2.5mm	67.102±30.696	56.722±15.494	0.069
	2.5~3.5mm	84.915±10.659	55.494±27.412	0.000
	3.5~4.5mm	63.825±6.575	57.990±15.890	0.380
	4.5~6mm	65.948±7.731	61.190±16.688	0.114
	Mean	69.108±5.854	58.796±10.447	0.000
OR	0~1.5mm	0.000±0.000	68.069±47.118	0.000
	1.5~2.5mm	-	130.505±95.987	0.000
	2.5~3.5mm	-	269.684±156.672	0.000
	3.5~4.5mm	-	130.807±105.306	0.000
	4.5~6mm	-	67.118±63.748	0.000
	Mean	-	133.237±70.990	0.000
GC	Minimum	77.667±4.984	30.640±24.854	0.000
	Mean	81.750±4.162	57.740±24.891	0.000
RNFL	Mean	95.708±8.143	73.900±21.461	0.000

nerve is mainly distributed. This implies that DCME may produce substantial degeneration in the central region of the macula and is closely related to blindness.

V. CONCLUSION

The results suggest that the proposed algorithm can reliably differentiate DCME patients from normal subjects. In addition, changes in the thickness of the IR and the POS may be useful risk factors for diagnosing DCME. Further, the most significant degeneration was observed in the central area of the macula as DCME progressed. These results suggest the clinical applicability of the proposed method to the diagnosis of DCME. This study has a limitation in that the normal and DCME groups had relatively small numbers of subjects. For more reliable results, future research should provide additional experiments using a larger number of test subjects and a wider range of classification models. In addition, the effects of age and gender should be considered in the context of DCME.

ACKNOWLEDGMENT

This work was supported by International Collaborative R&D Program funded by the Ministry of Knowledge Economy (MKE), Korea. (N01150049, Developing high frequency bandwidth [40-60 MHz] high resolution image system and probe technology for diagnosing cardiovascular lesion)

REFERENCES

[1] R. Klein, B. E. Klein, S. E. Moss, M. D. Davis, and D. L. DeMets, “The Wisconsin epidemiologic study of diabetic retinopathy. IV. Diabetic macular edema,” Ophthalmology, vol. 91, no. 12, Dec. 1984, pp. 1464-1474.

[2] B. E. Bouma, G. J. Tearney, and B. Bouma, “Handbook of optical coherence tomography,” Taylor & Francis, Marcel Dekker, Inc., New York, 2002.

[3] Y. Oshima, K. Emi, S. Yamanishi, and M. Motokura, “Quantitative assessment of macular thickness in normal subjects and patients with diabetic retinopathy by scanning

retinal thickness analyser," British Journal of Ophthalmology, vol. 83, no. 1, Jan. 1999, pp. 54-61.

[4] H. J. Zambarakji, W. M. Amoaku, and S. A. Vernon, "Volumetric analysis of early macular edema with the heidelberg retina tomograph in diabetic retinopathy," Ophthalmology, vol. 105, no. 6, Jun. 1998, pp. 1051-1059.

[5] M. R. Hee et al., "Quantitative assessment of macular edema with optical coherence tomography," Archieves of Ophthalmology, vol. 113, no. 8, Aug. 1995, pp. 1019-1029.

[6] D. F. Kiernan, W. F. Mieler, and S. M. Hariprasad, "Spectral-domain optical coherence tomography: a comparison of modern high-resolution retinal imaging systems," American Journal of Ophthalmology, vol. 149, no. 1, Jan. 2010, pp. 18-31.

[7] F. A. Medeiros et al., "Evaluation of retinal nerve fiber layer, optic nerve head, and macular thickness measurements for glaucoma detection using optical coherence tomography," American Journal of Ophthalmology, vol. 139, no. 1, Jan. 2005, pp. 44-55.

[8] M. Mujat et al., "Retinal nerve fiber layer thickness map determined from optical coherence tomography images," Optics Express, vol. 13, no. 23, Nov. 2005, pp. 9480-9491.

[9] D. Koozekanani, K. Boyer, and C. Roberts, "Retinal thickness measurements from optical coherence tomography using a Markov boundary model," IEEE Transactions on Medical Imaging, vol. 20, no. 9, Sep. 2001, pp. 900-916.

[10] P. Zhou and D. Pycock, "Robust statistical models for cell image interpretation," Image and Vision Computing, vol. 15, no. 4, Apr. 1997, pp. 307-316.

[11] D. U. G. Bartsch, X. Gong, C. Ly, and W. R. Freeman, "Optical coherence tomography: interpretation artifacts and new algorithm," Proc. SPIE 5370, Medical Imaging 2004: Image Processing, May 2004, pp. 2140-2151.

[12] A. Yazdanpanah, G. Hamarneh, B. Smith, and M. Sarunic, "Intra-retinal layer segmentation in optical coherence tomography using an active contour approach," Lecture Notes in Computer Science, vol. 5762, Sep. 2009, pp. 649-656.

[13] R. C. Gonzalez and R. E. Woods, "Digital image processing (3rd edition)," Prentice Hall, 2008.

[14] S. J. Chiu et al., "Automatic segmentation of seven retinal layers in SDOCT images congruent with expert manual segmentation," Optics Express, vol. 18, no. 18, Aug. 2010, pp. 19413-19428.

[15] Q. Yang et al., "Automated layer segmentation of macular OCT images using dual-scale gradient information," Optics Express, vol. 18, no. 20, Sep. 2010, pp. 21293-21307.

[16] R. Koprowski and Z. Wróbel, "Image processing in optical coherence tomography," Katowice, Poland, 2011.

[17] H. Ishikawa et al., "Macular segmentation with optical coherence tomography," Investigative Ophthalmology & Visual Science, vol. 46, no. 6, Jun. 2005, pp. 2012-2017.

[18] J. C. Brown et al., "Detection of diabetic foveal edema: contact lens biomicroscopy compared with optical coherence tomography," Archieves of Ophthalmology, vol. 122, no. 3, Mar. 2004, pp. 330-335.

 ISBN: 978-1-61208-367-4

Identifying Requirements for Centralized Service for Movement and Biodiversity Data Analysis

Ivana Nižetić Kosović, Boris Milašinović, Krešimir Fertalj
Department of Applied Computing
Faculty of Electrical Engineering and Computing, University of Zagreb
Zagreb, Croatia
e-mails: {ivana.nizetic, boris.milasinovic, kresimir.fertalj}@fer.hr

***Abstract*—Positioning technology is lately widely used in many scientific fields to collect movement and biodiversity data for further analysis. That generates enormous amount of positions and tracking data and impose the need for developing new algorithms for analysis and prediction, which are managed in ever growing partial and incomplete software solutions. In this paper, we made a roadmap for development of centralized service-oriented software in which one could manage data about moving objects or plants, as well as spatial layers, contextual information and perform complex algorithms. We identified a set of interfaces for communication with users allowing data and algorithm manipulation. Furthermore, we proposed meta-model for storing data and algorithms in order to achieve adaptiveness and wide applicability. The proposed model establishes a baseline for a concrete implementation.**

Keywords-movement data; biodiversity; tracking; service-oriented software; meta-model.

I. INTRODUCTION

Widespread use of Global Positioning System (GPS) devices, smart phones and wireless communication devices induced the expansion of research on moving objects [1]-[3]. There is an increasing number of applications in which mobility plays an important role. Vehicles are monitored and analysed in the field of traffic management and control to predict driver's intentions or traffic congestions [4]-[7]. Mobile users' movement is analysed to assure fast access point availability [8][9]. In the field of behavioural ecology, wild animals are tracked in order to predict their migrations and predator-prey behaviour [10]-[12]. There is also a variety of location-based services provided to smart phone users such as museum or touristic attractions applications for tourists, hospital plan information for doctors and nurses, hotels, location-aware games and so on.

Ubiquitous tracking technology and various applications generate enormous amount of tracking data and impose the need for developing new algorithms for analysis and prediction. Furthermore, in the last decade, the need to enrich object's movement data with geographical and semantic information is recognized since raw trajectory data (positions and timestamps) are not sufficient to obtain meaningful movement patterns [3][13][14]. Inclusion of heterogeneous data (contextual information, environmental data) makes analysis and prediction even more complex and the need for explanation of movement and spatial data more indispensable. Similar problem exists not only in the field of moving objects but also in some other fields like botany [15]. Although plants do not move, the algorithms for analysis and prediction are still complex and yield different results.

Still, there is no unique software in which one could manage data about moving objects, contextual information and perform complex algorithms. Visualisation and algorithms are currently managed by partial or incomplete platform-dependent software which cannot fully satisfy researchers' needs. Moreover, existing applications are not adaptable to new (custom) algorithms. Furthermore, since mobile applications cannot perform complex calculations and manage voluminous data locally, the need for standardized service-oriented system is essential.

These problems are further elaborated in Section II in order to formulate a problem followed by the identification of typical algorithms and their inputs and outputs. Once identified, it would be shown that there is a common intersection between inputs and outputs which can lead to a solution presented in the Section III. After centralized service-oriented solution concept description, a set of interfaces is enumerated, followed by the meta-model for storing data and algorithms. The paper ends with a conclusion of a presented work and future work guidelines.

II. PROBLEM FORMULATION

In the field of moving objects, an increasing number of analysis and prediction algorithms are developing: data mining techniques to extract behavioural patterns from moving objects data [16][17], clustering algorithms to detect important places [13][18], prediction techniques to model and predict moving object's future location, such as neural networks, Markov models, and specific types of dynamic Bayesian networks, like Hidden Markov Models (HMM) or Kalman filter [5][19]-[22]. In recent years, considerable research has been devoted to mapping the flora distribution, spatial analysis, biodiversity analysis and prediction of

 ISBN: 978-1-61208-367-4

occurrence. Bedia et al. [23] presented variety of different algorithms applied to a particular geographic area. However, it is not guaranteed that a particular algorithm can equally be used for different regions of the world. Although there are significant differences between moving objects and (static) plants, some similarities exist that led us to identify main ideas and enumerate problems in order to propose an integrated software solution. Table I shows representative categories of algorithms and their inputs and outputs with some examples where results depend on algorithm being used.

Although aforementioned algorithms and their implementations are valuable, we listed their main drawbacks considering not only their performance accuracy but also their ease of use and adaptiveness. As it can be seen from Table I, each algorithm uses spatial data and a set of coordinates that represents object positions or species findings. Testing different algorithms is usually a work intensive task as algorithms use different input data format and produce output in different format. Moreover, findings data must be exported to the appropriate format prior to the use of an algorithm and output must be transformed back into user's format repeating the similar task for many algorithms. Moreover, existent software is:

- Partial, incomplete – e.g., ArcGIS can be used for calculation of probabilities distribution of species, Weka and IntelligentMiner for clustering or basic HMM modelling
- Local – desktop apps, rarely applets or web services
- Closed – with no possibility for adding/customizing algorithms, and inaccessible to ordinary people, e.g., mobile users, non-experts in certain field
- Too specialized – not general enough to encompass various needs and heterogeneous data, e.g., HMM is applied to many classes of moving objects but each model is specialized only for that class although they have common structure and performance
- Technically determined – platform limited, e.g., applets require additional software installations, Geographic Information System (GIS) software requires high capacity and performance,
- Lack of (customized) visualisation of results

Consequently, there is a need for centralized, interoperable, opened, adaptive (to data and to algorithms) and widely useful application that could be extended with additional algorithms and additional data necessary for a particular algorithm.

TABLE I. EXAMPLES OF COMMONLY USED ALGORITHMS IN MOVEMENT AND BIODIVERSITY DATA ANALYSIS

Algorithm Category	**Input**	**Output**	**Results depend on algorithm?**
Pattern discovery - construction	Sets of marked sequential positions (positions, timestamps and corresponding pattern), Algorithm parameters (list of patterns, initial probabilities)	HMM (states, transitions, transition and emission probabilities)	Yes (HMM, State-space model, Artificial neural network - ANN, ...)
Pattern discovery - usage	A set of sequential positions (positions and timestamps)	A set of marked sequential positions (positions, timestamps and corresponding pattern)	Yes (HMM, State-space model, ANN, ...)
Positions clustering	Positions Algorithm parameters (Eps, MinPt)	Sets of positions (clusters) and a set of noise positions	Yes (Density-based spatial clustering of applications with noise - DBSCAN, …)
Species distribution	Spatial layers, species findings	Matrix of species findings per spatial layer attribute	No
Species distribution prediction	Spatial layers, species findings	Matrix of probabilities corresponding to spatial shapes	Yes (Distance alg., Maximum Entropy Modelling (MAXENT), Multiple Logistic Regression - MLR, ANN,...)
Ecological profile	Spatial layers, species findings	Matrix of species occurrence per spatial layer attribute or spatial unit	No
Biodiversity analysis (alpha and research intensity)	Spatial layers, species findings	Collection of matrices (alpha and research intensity per layer attribute, other species data - ecological indices per spatial layer attribute, basic species data...) Extension of a spatial layer with data from matrices but for a unit instead of attribute	May depend, when data uncertainty exists [15]
Movement prediction	Positions, spatial layers, contextual data	Positions	Yes (ANN, HMM, ...)

 ISBN: 978-1-61208-367-4

Recently, some notable ideas of centralized solutions for managing moving objects' data and performing algorithms were proposed. Xu and Guting [24] proposed a generic data model for moving objects that can apply in more than one environment and applied it to transportation model. A conceptual data model for representing semantic trajectories applied to tourism and animal movement is shown by Bogorny et al. [25], giving the baseline for future research on semantic trajectory.

Considering wild life research there is a vision of centralized solution for wildlife data management in [26]. We have also presented a generic model and proposed the conceptual data model for analysis and movement prediction independent of application area and moving object type [27].

In [15], we proposed an object model for biodiversity analysis that can avoid the problem of data export. Input and output are modelled using interfaces thus making the model available in various usage scenarios as a web service or a layer in an application. As it only defines structure of input data, the model is independent of concrete data storage and the service is implemented in such way that it should be independent from data retrieval as long as the data follows some biological patterns. Data retrieval is done by implementing proposed interfaces and merging them with the core service implementation service using one of dependency injection techniques when the service is exposed as a web service.

An idea of an integrated solution is also presented by Ames et al. [28]. The authors have developed web services-based software for hydrologic data discovery, download, visualization, and analysis using extensible plug-ins for searching, viewing and exporting data.

Although from a different field of study, cited papers yielded an idea of using web services, data interfaces and plug-ins for algorithms. Using service interfaces and various plug-ins it would be easier to try a different type of analysis or analyze the same thing using different algorithms without need to do it manually or to convert input and output data between different formats.

III. A PROPOSED SOLUTION

A. *Concept of centralized service-oriented solution and its interfaces*

Figure 1 presents a conceptual view of centralized service-oriented solution we propose. The application consists of data stored permanently or temporary on a server, a set of algorithms that use data to produce result to be returned to a user and a set of interfaces for communication with users allowing data and algorithm manipulation. Each of the arrows from the figure presents usage of an interface, which we would only enumerate in the paper providing input and output descriptively leaving format to be standardized in the future work.

Positions are main input data to perform any algorithm. They can be collected via mobile device or uploaded using a personal computer but also can be stored by an on-premise server. As mentioned previously, one of disadvantages of existing software is that a user must export his/her data, adjust it to required formats and upload it again to a proprietary service/application. In the proposed system, the user can keep data on his/her server and enable a service on his/her own data server according to one of formats that would be proposed as a standard. Instead of concrete data, the user could enter service location and optional filters and data would be retrieved later as needed.

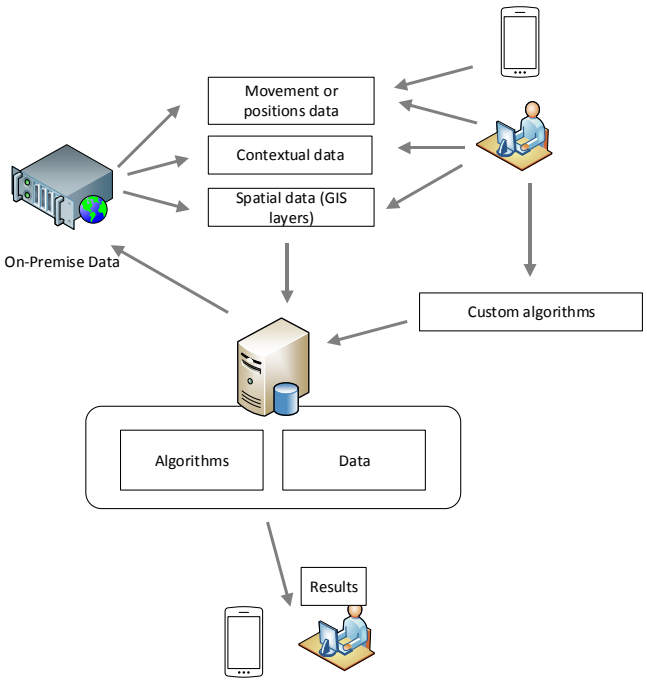


Figure 1. Concept of centralized service-oriented solution

The same principle applies also to contextual data and spatial data (GIS layers). Contextual data are usually environmental data, data about certain object, e.g., species etc. These data are collected from different sources: other data providing services, for example weather services [29][30] or they could be sent by user as well (for example, specific layer considering certain species). Contextual data are not limited in scope and additional information is facultative or obligatory depending on an algorithm requirements. As shown in Table I, some of analysis depends on used algorithm and a particular algorithm may need some additional data. Therefore these data can be of any type and format of data that can be recognized and used by a particular algorithm.

The idea is that algorithms are categorized and that there exists default algorithm per category but the application enables upload of a user's own algorithm. A solution must provide an API for access to movement and spatial data and any custom data from an input. An algorithm should implement an interface from a category it belongs to. Details, like language, format, runtime environment, etc., are left to implementation phase.

 ISBN: 978-1-61208-367-4

The results of algorithms can further be analysed and deployed to the server as well. Either retrieved on a temporary base or stored permanently, data stored on the server consists of data and algorithms which are both meta-modelled.

Summarizing all above, the set of interfaces used in Figure 1 can be described as in Table II. For each category of data, there are three possible interfaces: for data upload, for setting URL of a service that host user's data and for returning on-premise data once that centralized solution need users data.

B. Meta-model

To achieve adaptiveness and wide applicability, data and algorithms should be meta-modelled (Figure 2). Usage of a meta-model enables addition of algorithms, algorithms categories, domains and attributes that does not exist in the present moment but would occur in the future. Furthermore it enables integration of many different types of users under one, integrated model.

1) Object and its characteristics

Central entity of movement and biodiversity data is an *object* which represents either an animal or a plant. An object has recursive relationship to itself, thus allowing to model taxonomy (biological or zoological hierarchy of species) at any level - from unique instance of certain species to kingdom. For example, an object could be a certain animal (identified by name or a collar identifier), which is of species grey wolf (*canis lupus*), which is of genus *canis*, or it can be a species, e.g., *abies alba*, genus *abies* etc. Any of taxonomic ranks can have attributes which can comprise values from a certain domain. Values of defined attributes can be assigned to object itself. For example, species wolf can have attribute *social order*, while a particular wolf can have value of that attribute *alpha male*.

2) Findings (positions)

An object can be spotted at certain coordinate at certain timestamp. The attribute *source* represents type (source) of finding, for example GPS collar, terrain research, photo, literature citation etc. For example, a GPS collar carried by a wolf retrieves a recorded position of the wolf at a certain timestamp. Also, a researcher could see a wolf at a terrain and also record time and coordinates of finding. Another example is logging a plant's position by a researcher at the terrain or entering coordinates and timestamp of the plant from literature or a photo taken at the terrain.

Similarly to *time* and *source* other contextual data for a finding could be stored in the model. Optionally, if necessary, these contextual data can be related to attributes and domains.

3) Spatial data

Since spatial data is present in almost every kind of static or moving objects' positions analysis, they are meta-modelled as well. A layer is consisted of spatial elements (shapes) and layer attributes. By the term layer, we mean a GIS layer, such as a content of ESRI *shapefile* [31]. The shapes (usually polygons) belong to a certain layer. Values of defined attributes, which are valid in specific period of time, are assigned to certain shape. A coordinate (at which the object is spotted) belongs to certain shapes.

TABLE II. LIST OF INTERFACES

Category	Purpose(s)	Input	Output
Movement and positions	Data upload	List of n-tuples containing coordinates, species or object identifier and identifier of additional context data	
	Provides URL of a service hosted on a user's server (on-premise data)	URL of a service and custom data that should be send as-is to the service	On-premise server produce output same as used for input when data uploaded directly
Contextual data	Upload of species/objects data	List of n-tuples containing species/object identifier and additional parameters of any type (e.g., byte array) that must be interpreted by an algorithm	
	Upload of contextual information, i.e., those that are related to positions	List of n-tuples containing context identifier and additional parameters of any type (e.g., byte array) that must be interpreted by an algorithm	
	Provides URL of a service hosted on a user's server (on-premise data)	URL of a service for contextual data	On-premise server produce output same as used for input when data uploaded directly
Spatial data (GIS layers)	Data upload	Zipped folder containing one of supported GIS formats (e.g., ESRI shp+dbf+shx)	
	Provides URL of a services hosted on a user's server (on-premise data)	URL of a service that serves a GIS layer (e.g., WMS server)	Zipped folder or WMS server
Custom algorithms	Custom algorithm upload	Algorithm code	
Results	Results of an algorithm		One or more matrices that summarize algorithm results and optional output from custom algorithm that must be interpreted by a user
Analysis request	Initiate analysis	Type of analysis, chosen algorithm, optional filter on context and data	

 ISBN: 978-1-61208-367-4

4) Algorithms

Several classes of algorithms used in movement and biodiversity analysis are summarized in Table II. In each category, input and output types can be identified and these input and output parameters can be meta-modelled using entity *Attribute* thus describing interfaces for all algorithms in the category. In order to enable custom extension of parameters, the last parameter should always be in free form (e.g., XML or array of bytes) for custom data.

One or more algorithms can exist for each category and one of them is default one for the category. An algorithm can have additional parameters needed for a concrete implementation. Each algorithm instance produces results using data from findings, coordinates and shapes which do not have to be modelled as a relation in a model, but should be available using some form of API in implementation. Results from an algorithm instance are stored according to output parameters attributes.

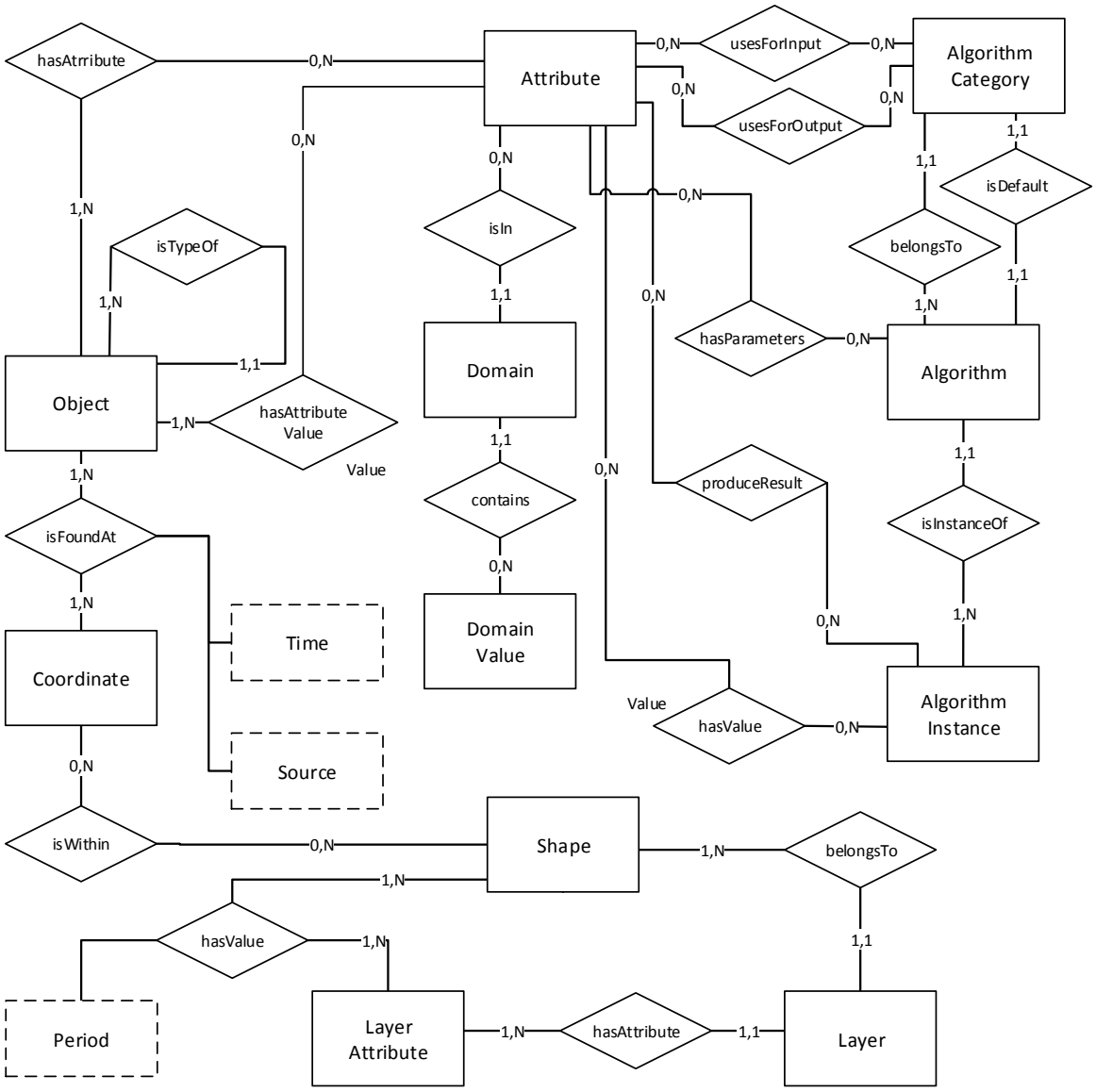


Figure 2. Meta-model for movement and biodiversity data and algorithms

IV. CONCLUSION AND FUTURE WORK

The proposed solution and the designed model establish a baseline for a concrete implementation. To achieve interoperability and openness, the proposed solution should be exposed as (web) services with clearly defined standards of input/output data. Future work should be related to the standardization of input and output formats. Furthermore, an appropriate solution for writing algorithms must be identified (in place compilation of a code in some specific language or custom Domain Specific Language) and APIs for data access must be defined.

REFERENCES

[1] S. Dodge, R. Weibel, and A.-K. Lautensch, "Towards a taxonomy of movement patterns," Information Visualization, vol. 7, 2008, pp. 240-252

[2] J. Drummond, R. Billen, E. Joao, and D. Forrest, Dynamic and Mobile GIS: Investigating Changes in Space and Time, CRC Press, Taylor & Francis Inc, 2006.

[3] C. Parent, et al., "Semantic trajectories modeling and analysis," ACM Comput. Surv., vol. 45, 2013, pp. 1-32

[4] W. Wang and G. Wets, Computational Intelligence for Traffic and Mobility: Atlantis Publishing Corporation, 2013.

[5] C. Barrios and Y. Motai, "Improving Estimation of Vehicle's Trajectory Using the Latest Global Positioning System With Kalman Filtering," IEEE T. Instrumentation and Measurement, vol. 60, 2011, pp. 3747-3755

[6] J. Krumm and E. Horvitz, "Predestination: Inferring Destinations from Partial Trajectories," UbiComp 2006: Ubiquitous Computing, vol. 4206, 2006, pp. 243-260

[7] J. Froehlich and J. Krumm, "Route Prediction from Trip Observations," presented at the Society of Automotive Engineers (SAE) 2008 World Congress, 2008.

[8] J.-M. François, G. Leduc, and S. Martin, "Learning movement patterns in mobile networks: a generic method," presented at the European Wireless 2004, Barcelona, Spain, 2004.

[9] G. Yavas, D. Katsaros, Ö. Ulusoy, and Y. Manolopoulos, "A data mining approach for location prediction in mobile environments," Data Knowl. Eng., vol. 54, 2005, pp. 121-146

[10] A. Franke, T. Caelli, G. Kuzyk, and R. J. Hudson, "Prediction of wolf (Canis lupus) kill-sites using hidden Markov models," Ecological Modelling, vol. 197, Aug 10 2006, pp. 237-246

[11] N. F. Webb, M. Hebblewhite, and E. H. Merrill, "Statistical methods for identifying wolf kill sites using global positioning system locations," Journal of Wildlife Management, vol. 72, Apr 2008, pp. 798-807

[12] A. D. Middleton, et al., Linking anti-predator behaviour to prey demography reveals limited risk effects of an actively hunting large carnivore, 2013, pp. 1023–1030

[13] L. O. Alvares, et al., "A model for enriching trajectories with semantic geographical information," presented at the Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems, Seattle, Washington, 2007.

[14] B. Guc, M. May, Y. Saygin, and C. Körner, "Semantic Annotation of GPS Trajectories " presented at the 11th AGILE International Conference on Geographic Information Science 2008, University of Girona, Spain, 2008.

[15] B. Milašinović, T. Nikolić, and K. Fertalj, "Biodiversity analysis supporting species-subspecies uncertainty in findings data," International journal of biology and biomedical engineering, vol. 7, 2013, pp. 127-134

[16] Y. Ishikawa, Data Mining for Moving Object Databases. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2010.

[17] S. Elnekave, M. Last, and O. Maimon, "Measuring Similarity Between Trajectories of Mobile Objects," in Applied Pattern Recognition. vol. 91, H. Bunke, et al., Eds., ed: Springer Berlin / Heidelberg, 2008, pp. 101-128.

[18] I. Nižetić Kosović, K. Fertalj, and J. Kusak, "An expert system for discovering biogeographically interesting locations from animal movement data," presented at the European Computing Conference, Paris, France, 2011, pp. 348-353

[19] S. Akoush and A. Sameh, "Movement Prediction Using Bayesian Learning for Neural Networks," presented at the International conference on Wireless communications and mobile computing, Honolulu, Hawaii, USA, 2007, pp. 191-196

[20] R. Joo, S. Bertrand, J. Tam, and R. Fablet, "Hidden Markov Models: The Best Models for Forager Movements?," PLoS ONE, vol. 8 , 2013,

[21] P. E. Smouse, et al., "Stochastic modelling of animal movement," Philosophical Transactions of the Royal Society B-Biological Sciences, vol. 365, Jul 27 2010, pp. 2201-2211

[22] I. Nižetić, K. Fertalj, and D. Kalpić, "A Prototype for the Short-term Prediction of Moving Object's Movement using Markov Chains," presented at the 31st International Conference on Information Technology Interfaces, Cavtat, Croatia, 2009, pp. 559 - 564

[23] J. Bedia, J. Busqué, and J. M. Gutiérrez, "Predicting plant species distribution across an alpine rangeland in northern Spain. A comparison of probabilistic methods," Applied Vegetation Science, vol. 14, 2011, pp. 415-432

[24] J. Xu and R. H. Guting, "A generic data model for moving objects," Geoinformatica, vol. 17, 2013, pp. 125-172

[25] V. Bogorny, C. Renso, A. R. d. Aquino, F. d. L. Siqueira, and L. O. Alvares, "CONSTAnT – A Conceptual Data Model for Semantic Trajectories of Moving Objects," Transactions in GIS, vol. 18, 2014, pp. 66-88

[26] F. Urbano, et al., "Wildlife tracking data management: a new vision," Philosophical Transactions of the Royal Society B-Biological Sciences, vol. 365, Jul 27 2010, pp. 2177-2185

[27] I. Nižetić and K. Fertalj, "Automation of the Moving Objects Movement Prediction Process Independent of the Application Area," Computer Science and Information Systems, vol. 7, 2010, pp. 931-945

[28] D. P. Ames, et al., "HydroDesktop: Web services-based software for hydrologic data discovery, download, visualization, and analysis," Environmental Modelling & Software, vol. 37, 2012, , pp. 146-156

[29] M. Kanamitsu, et al., "NCEP-DEO AMIP-II Reanalysis (R-2)," Bulletin of the American Meteorological Society, vol. 83, 2002, pp. 1631-1643

[30] NOAA. (2011, 01.06.2011). Earth System Research Laboratory. Available: http://www.esrl.noaa.gov/

[31] ESRI. *ESRI Shapefile Technical Description. An ESRI White Paper—July 1998* [Online]. Available from: http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf 2014.08.15

 ISBN: 978-1-61208-367-4

Method for Analytic Evaluation of the Weights of a Robust Large-Scale Multilayer Neural Network with Many Hidden Nodes

Mikael Fridenfalk
Department of Game Design
Uppsala University
Visby, Sweden
mikael.fridenfalk@speldesign.uu.se

***Abstract*—The multilayer feedforward neural network is presently one of the most popular computational methods in computer science. The current method for the evaluation of its weights is however performed by a relatively slow iterative method known as backpropagation. According to previous research, attempts to evaluate the weights analytically by the linear least square method, showed to accelerate the evaluation process significantly. The evaluated networks showed however to fail in robustness tests compared to well-trained networks by backpropagation, thus resembling overtrained networks. This paper presents the design and verification of a new method, that solves the robustness issues for a large-scale neural network with many hidden nodes, as an upgrade to the previously suggested analytic method.**

Keywords-analytic; FNN; large-scale; least square method; neural network; robust; sigmoid

I. INTRODUCTION

The artificial neural network constitutes one of the most interesting and popular computational methods in computer science. The most well-known category is the multilayer Feedforward Neural Network (FNN), where the weights are estimated by an iterative training method called backpropagation [7][9]. Although this iterative method is relatively fast for small networks, it is rather slow for large ones, given the computational power of modern computers [1][2]. To accelerate the training speed of FNNs, many approaches have been suggested based on the least square method [3]. Although the presentation on the implementation, as well as of the data on the robustness of these methods may be improved, the application of the least square method as such seems to be a promising path to investigate [8][10].

What we presume to be required for a new method to replace backpropagation in such networks, is not only that it is efficient, but also that it is superior compared to existing methods and is easy to understand and implement. The goal of this paper is, therefore, to investigate the possibility to find a *robust analytic solution* (i.e., with good generalization abilities compared with a well-trained network using backpropagation, but without any iterations involved), for the weights of an FNN, that is easily understood and that may be implemented relatively effortlessly, using a mathematical application such as Matlab [6].

In a previous work [4], an analytic solution was proposed for the evaluation of the weights of a textbook FNN. This solution was found to be significantly much faster, and for $H = N - 1$ (where H denotes the number of hidden nodes and N, the number of training points), more accurate than solutions provided by backpropagation, but at the same time significantly less robust compared with a well-trained network using backpropagation, why the analytic solution was considered to lack robustness for direct use.

Further experiments showed, however, that even small measures, such as an increase in the input range of the network by doubling the size of the training set with the addition of

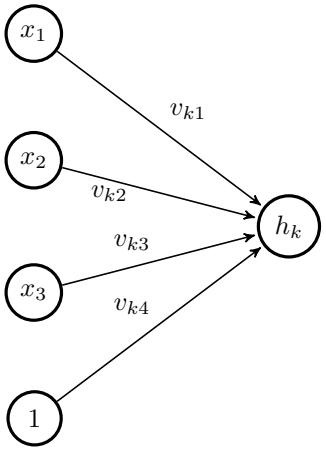


Figure 1. An example with three input nodes ($M = 3$), $h_k = S(\mathbf{v}_k\mathbf{u}) = S(v_{k1}x_1 + v_{k2}x_2 + v_{k3}x_3 + v_{k4})$, using a sigmoid activation function S.

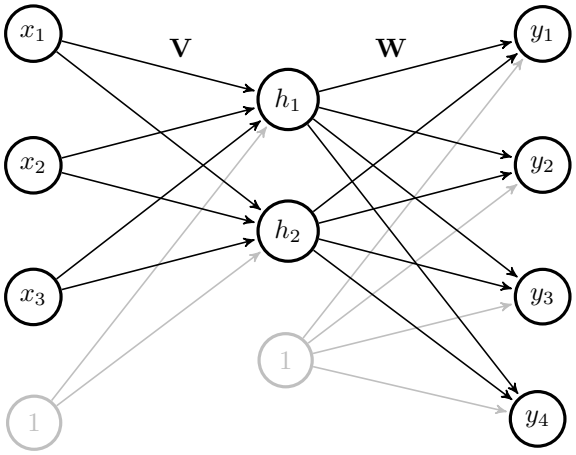


Figure 2. A vectorized model of a standard FNN with a single hidden layer, in this example with $M = 3$ input nodes, $H = 2$ hidden nodes, $K = 4$ output nodes and the weight matrices $\mathbf{V}$ and $\mathbf{W}$, using a sigmoid activation function for the output of each hidden node. In this model, the biases for the hidden layer and the output layer correspond to column $M + 1$ in $\mathbf{V}$ versus column $H + 1$ in $\mathbf{W}$.

 ISBN: 978-1-61208-367-4

perturbation, led to significant improvement of the robustness of the evaluated network. As a first systematic attempt to address the issue of robustness, this paper presents the derivation, implementation and verification of a new method, based on the expansion of the training set of an FNN, with addition of perturbation, but in practice without any impact on the execution speed of the original method. As a brief overview, in Section II, a recap is made of the theory behind the analytic method presented in [4], which is the foundation of the theory presented in this paper. In Section III, a new method (or upgrade) is derived for the improvement of the robustness of the original method. In Section IV, the experimental setup is briefly described, and in Section V, the new method is experimentally verified by comparison with the performance of the original one.

II. Analytic Solution

In [4], a textbook FNN is vectorized based on a sigmoid activation function $S(t) = 1/(1 + e^{-t})$. The weights $\mathbf{V}$ and $\mathbf{W}$ of such system (often denoted as W_{IH} versus W_{HO}), may be represented by Figures 1-2. In this representation, defined here as the normal form, the output of the network may be expressed as:

$$\mathbf{y} = \mathbf{W}\mathbf{h} = \mathbf{W}\begin{bmatrix} S(\mathbf{V}\mathbf{u}) \\ \hline 1 \end{bmatrix}, \ \mathbf{u} = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} \tag{1}$$

where $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_M]^T$ denotes the input signals, $\mathbf{y} = [y_1 \ y_2 \ \dots \ y_K]^T$ the output signals, and S, an element-wise sigmoid function. In this paper, a winner-take-all classification model is used, where the final output of the network is the selection of the output node that has the highest value. Since the sigmoid function is constantly increasing and identical for each output node, it can be omitted from the output layer, as $\max(\mathbf{y})$ results in the same node selection as $\max(S(\mathbf{y}))$. Further on, presuming that the training set is highly fragmented (the input-output relations in the training sets were in our experiments established by a random number generator), the number of hidden nodes is preferred to be set to $H = N - 1$. Defining a batch (training set), the input matrix $\mathbf{U}$, may be expressed as:

$$\mathbf{U} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{M1} & x_{M2} & \cdots & x_{MN} \\ 1 & 1 & \cdots & 1 \end{bmatrix} \tag{2}$$

where column vector i in $\mathbf{U}$, corresponds to training point i, column vector i in $\mathbf{Y}_0$ (target output value) and in $\mathbf{Y}$ (actual output value). Further, defining $\mathbf{H}$ of size $N \times N$, as the batch values for the hidden layer, given a training set of input and output values and $M^+ = M + 1$, the following relations hold:

$$\mathbf{U} = \begin{bmatrix} \mathbf{X} \\ \hline \mathbf{1}^T \end{bmatrix} : [M^+ \times N] \tag{3}$$

$$\mathbf{H} = \begin{bmatrix} S(\mathbf{V}\mathbf{U}) \\ \hline \mathbf{1}^T \end{bmatrix} : [N \times N] \tag{4}$$

$$\mathbf{Y} = \mathbf{W}\mathbf{H} : [K \times N] \tag{5}$$

To evaluate the weights of this network analytically, we need to evaluate the target values (points) of $\mathbf{H}_0$ for the hidden layer. In this context, the initial assumption is that any point is feasible, as long as it is unique for each training set. Therefore, in this model, $\mathbf{H}_0$ is merely composed of random numbers. Thus, the following evaluation scheme is suggested for the analytic solution of the weights of such network:

$$\mathbf{V}^T = (\mathbf{U}\mathbf{U}^T)^{-1}\mathbf{U}\mathbf{H}_0^T : [M^+ \times H] \tag{6}$$

$$\mathbf{W}^T = (\mathbf{H}\mathbf{H}^T)^{-1}\mathbf{H}\mathbf{Y}_0^T : [N \times K] \tag{7}$$

where a least square solution is used for the evaluation of each network weight matrix. Such equation is nominally expressed as:

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{8}$$

with the least square solution [3]:

$$\mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b} \tag{9}$$

Since the mathematical expressions for the analytic solution of the weights of a neural network may be difficult to follow, an attempt has been made in Figure 3 to visualize the matrix operations involved. While a nonlinear activation function (such as the sigmoid function) is vital for the success of such network, the inclusion of a bias is not essential. It is for instance possible to omit the biases and to replace $\mathbf{H}_0$ with an identity matrix $\mathbf{I}$. Such a configuration would instead yield the following formula for the evaluation of $\mathbf{V}$ and $\mathbf{H}$ (where $\mathbf{U}\mathbf{I}$ can further be simplified as $\mathbf{U}$):

$$\mathbf{V}^T = (\mathbf{U}\mathbf{U}^T)^{-1}\mathbf{U}\mathbf{I} : [M^+ \times N] \tag{10}$$

$$\mathbf{H} = S(\mathbf{V}\mathbf{U}) : [H \times N] \tag{11}$$

III. Proposal

To start with, we expand the input training set $\mathbf{U}$ in (2), by the addition of perturbation to the input signal, given the definition $\mathbf{\Theta} = \mathbf{U}\mathbf{U}^T$, with $\theta_{M^+M^+} = N$ in:

$$\mathbf{\Theta} = \begin{bmatrix} \theta_{11} & \theta_{12} & \dots & \theta_{1M} & \theta_{1M^+} \\ \theta_{21} & \theta_{22} & \dots & \theta_{2M} & \theta_{2M^+} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \theta_{M1} & \theta_{M2} & \dots & \theta_{MM} & \theta_{MM^+} \\ \theta_{M^+1} & \theta_{M^+2} & \dots & \theta_{M^+M} & N \end{bmatrix} \tag{12}$$

Further, an extended matrix $\tilde{\mathbf{U}}$ is introduced, where:

$$\tilde{\mathbf{U}} = \begin{bmatrix} \tilde{\mathbf{U}}_1 & \tilde{\mathbf{U}}_2 & \dots & \tilde{\mathbf{U}}_N \end{bmatrix} \tag{13}$$

with:

$$\mathbf{U}_j = \begin{bmatrix} u_{1j} + \Delta & u_{1j} - \Delta & u_{1j} & u_{1j} \\ u_{2j} & u_{2j} & u_{2j} + \Delta & u_{2j} - \Delta \\ \vdots & \vdots & \vdots & \vdots \\ u_{Mj} & u_{Mj} & u_{Mj} & u_{Mj} \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{matrix} \cdots & u_{1j} & u_{1j} \\ \cdots & u_{2j} & u_{2j} \\ \ddots & \vdots & \vdots \\ \cdots & u_{Mj} + \Delta & u_{Mj} - \Delta \\ \cdots & 1 & 1 \end{matrix} \tag{14}$$

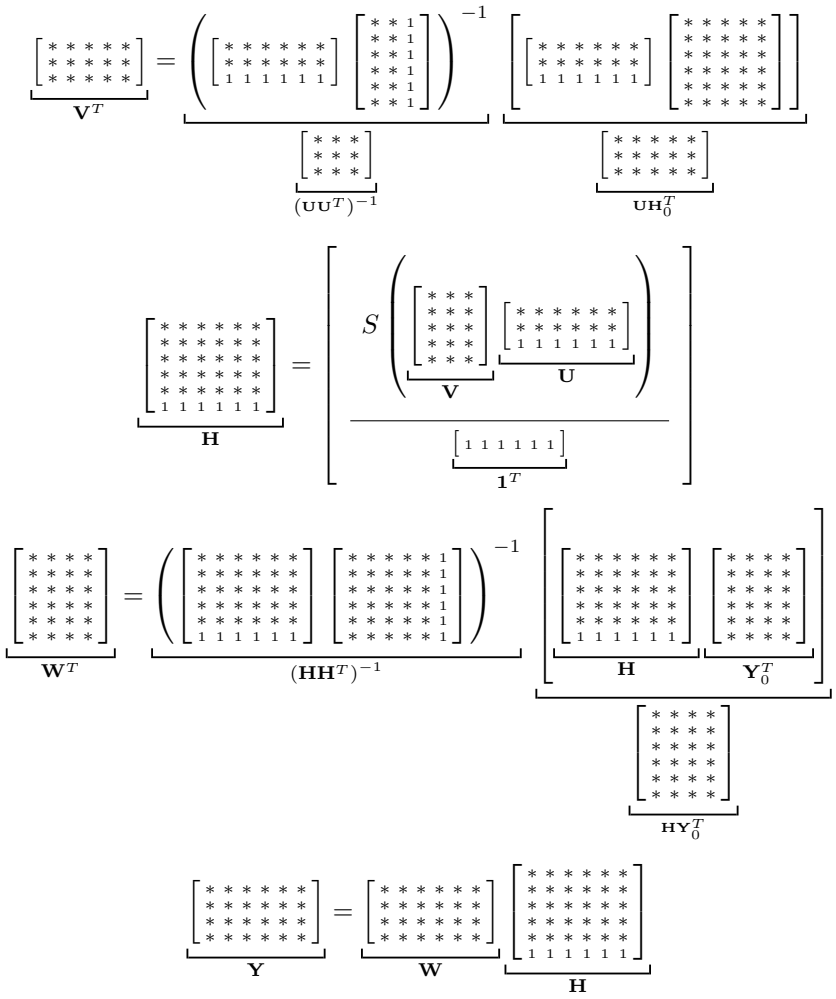


Figure 3. A visual representation of the evaluation of weights $\mathbf{V}$ and $\mathbf{W}$ by the analytic method presented in the related work [4], and the actual output $\mathbf{Y}$, in this example as a function of six training points, $N = 6$, the training input and output sets $\mathbf{U}$ and $\mathbf{Y}_0$, with two inputs, $M = 2$, four outputs, $K = 4$, and five hidden nodes, $H = N - 1 = 5$. In this figure, an asterisk denotes a floating-point number. To facilitate bias values, certain matrix elements are set to one.

and where Δ is defined as the amplitude of the perturbation. Thus, for the right hand side of (8), $\tilde{\mathbf{\Theta}} = \tilde{\mathbf{U}}\tilde{\mathbf{U}}^T$, or more explicitly:

$$\tilde{\mathbf{\Theta}} = 2M \begin{bmatrix} d_1 & \theta_{12} & \dots & \theta_{1M} & \theta_{1M^+} \\ \theta_{21} & d_2 & \dots & \theta_{2M} & \theta_{2M^+} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \theta_{M1} & \theta_{M2} & \dots & d_M & \theta_{MM^+} \\ \theta_{M^+1} & \theta_{M^+2} & \dots & \theta_{M^+M} & N \end{bmatrix} \tag{15}$$

with $d_i = \theta_{ii} + \alpha$, where $\alpha = N\Delta^2/M$, or:

$$\tilde{\mathbf{\Theta}} = 2M \left[\mathbf{\Theta} + \text{diag}(\alpha, \alpha, \dots, \alpha, 0)\right] \tag{16}$$

where $\text{diag}(d_1, d_2, \dots, d_{M^+})$, denotes a diagonal matrix of size $M^+ \times M^+$ (where $M^+ = M + 1$), with the diagonal elements $d_1, d_2, \dots, d_{M^+}$. Similarly, for the left hand side of (8), $\mathbf{\Psi}$ and $\mathbf{\Lambda}$ are defined as:

$$\mathbf{H}_0^T = \mathbf{\Psi} = \begin{bmatrix} \psi_{11} & \psi_{12} & \dots & \psi_{1H} \\ \psi_{21} & \psi_{22} & \dots & \psi_{2H} \\ \vdots & \vdots & \ddots & \vdots \\ \psi_{N1} & \psi_{N2} & \dots & \psi_{NH} \end{bmatrix} \tag{17}$$

$$\mathbf{\Lambda} = \mathbf{U}\mathbf{H}_0^T = \begin{bmatrix} \lambda_{11} & \lambda_{12} & \dots & \lambda_{1H} \\ \lambda_{21} & \lambda_{22} & \dots & \lambda_{2H} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{M+1} & \lambda_{M+2} & \dots & \lambda_{M+H} \end{bmatrix} \tag{18}$$

and thereby, $\mathbf{\Psi}$ and $\mathbf{\Psi}_j$ as:

$$\mathbf{\Psi} = \begin{bmatrix} \mathbf{\Psi}_1 \\ \mathbf{\Psi}_2 \\ \vdots \\ \mathbf{\Psi}_N \end{bmatrix} \tag{19}$$

$$\mathbf{\Psi}_j = \begin{bmatrix} \psi_{j1} & \psi_{j2} & \dots & \psi_{jH} \\ \psi_{j1} & \psi_{j2} & \dots & \psi_{jH} \\ \vdots & \vdots & \ddots & \vdots \\ \psi_{j1} & \psi_{j2} & \dots & \psi_{jH} \end{bmatrix} \tag{20}$$

with $\tilde{\mathbf{\Lambda}} = \tilde{\mathbf{U}}\mathbf{\Psi}$:

$$\tilde{\mathbf{\Lambda}} = 2M\mathbf{\Lambda} \tag{21}$$

This transforms (8) into:

$$\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T\mathbf{X} = \tilde{\mathbf{U}}\mathbf{\Psi} \tag{22}$$

 ISBN: 978-1-61208-367-4

or:

$$\tilde{\mathbf{\Theta}}\mathbf{X} = \tilde{\mathbf{\Lambda}} \tag{23}$$

Thus:

$$2M\left[\mathbf{\Theta} + \text{diag}(\alpha, \alpha, \ldots, \alpha, 0)\right]\mathbf{X} = 2M\mathbf{\Lambda} \tag{24}$$

Given the matrix equation:

$$\mathbf{AX} = \mathbf{B} \tag{25}$$

since, given a scalar $c \in \mathbb{R}$:

$$c \cdot (\mathbf{AX}) = (c \cdot \mathbf{A})\mathbf{X} = c \cdot \mathbf{B} \tag{26}$$

thereby:

$$\left[\mathbf{\Theta} + \text{diag}(\alpha, \alpha, \ldots, \alpha, 0)\right]\mathbf{X} = \mathbf{\Lambda} \tag{27}$$

This yields thus, the final expression:

$$\left[\mathbf{U}\mathbf{U}^T + \text{diag}(\alpha, \alpha, \ldots, \alpha, 0)\right]\mathbf{X} = \mathbf{U}\mathbf{H}_0^T \tag{28}$$

Hence, the expansion of $\mathbf{U}$ into a perturbation matrix $\tilde{\mathbf{U}}$ of size $M^+ \times 2MN$, and similarly of $\mathbf{H}_0^T$ into a matrix $\mathbf{\Psi}$ of size $2MN \times H$, is according to (28), equivalent to the reinforcement of the diagonal elements of the square matrix $\mathbf{\Theta} = \mathbf{U}\mathbf{U}^T$, by the addition of a factor $\alpha = N\Delta^2/M$ to each diagonal element, except for the last one, which as a consequence of the use of bias in the network, is left intact.

IV. Experimental Setup

The experiments presented in this paper are based on a minimal mathematical engine that was developed in C++, with the capability to solve $\mathbf{X}$ in a linear matrix equation system of the form $\mathbf{AX} = \mathbf{B}$, where $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{X}$ denote matrices of appropriate sizes, since it is computationally more efficient to solve a linear equation system directly, than by matrix inversion. In this system, the column vectors of $\mathbf{X}$ are evaluated using a single Gauss-Jordan elimination cycle [3], where each column vector $\mathbf{x}_i$ in $\mathbf{X}$ corresponds to the column vector $\mathbf{b}_i$ in $\mathbf{B}$. Backpropagation was in these experiments, for high execution speed (and a fair comparison with the new methods), also implemented in C++, using the code presented in [5] as a reference.

V. Results

The experimental results presented in this paper are shown in Figures 4-9, measuring average success rate, and Table I, measuring execution speed. Each experiment is based on ten individual experiments (with different random seeds), using a single CPU-core on a modern laptop computer. In Table I, $\bar{t}_{\text{bp}}$ denotes the execution time for backpropagation based on 10000 iterations, which applies to all backpropagation experiments presented in this paper. Similarly, $\bar{t}_{\text{new}}$ denotes the execution time for the original analytic method in [4], and $\bar{t}_{\text{new+}}$, the execution time for the new method presented in this paper, using diagonal reinforcement.

In these experiments, the input values to the FNN consisted of the integers $\{0, 1, 2\}$, and the output values of a binary number, $\{0, 1\}$. To avoid inconsistencies (or repetition) in any training set, identical input values were replaced by unique values. For the addition of noise, a random value (with uniform distribution) in the range of $\pm\Delta$, was added to each input value. However, although according to our derivation of (28), $\alpha = N\Delta^2/M$, α had in practice to be retuned to $10^5 \cdot N\Delta^2$ for good results in the experiments in Figures 4-6 ($H = N - 1$), and to $10^4 \cdot N\Delta^2$ in Figures 7-9 (few hidden nodes).

TABLE I. Average execution time

Figure	M	N	H	K	$\bar{t}_{\text{bp}}$	$\bar{t}_{\text{new}}$	$\bar{t}_{\text{new+}}$
4	10	25	24	10	92.5 ms	688 μs	668 μs
5	20	50	49	20	332 ms	4.84 ms	4.86 ms
6	40	100	99	40	1.27 s	37.0 ms	37.1 ms
7	10	25	10	10	43.3 ms	212 μs	202 μs
8	20	50	20	20	144 ms	1.33 ms	1.31 ms
9	40	100	40	40	535 ms	9.35 ms	9.38 ms

VI. Conclusion

The upgrade proposed in this paper, showed to solve the robustness issues of the analytic solution of the weights of a large-scale FNN with $H = N - 1$ nodes, and in practice without any impact on the execution speed of the solution. Since according to [4], the original method was not considered to be ready for direct use until the robustness issues had been solved, this upgrade provides hereby a method that, given access to a linear equation solver, while considerably faster, is for large-scale networks with many hidden nodes, comparable in robustness to a well-trained FNN by backpropagation.

References

[1] P. De Wilde, Neural Networks Models: An Analysis, Springer, 1996, pp. 35-51.

[2] R. P. W. Duin, "Learned from Neural Networks", ASCI2000, Lommel, Belgium, 2000, pp. 9-13.

[3] C. H. Edwards and D. E. Penney, Elementary Linear Algebra, Prentice Hall, 1988, pp. 220-227.

[4] M. Fridenfalk, "The Development and Analysis of Analytic Method as Alternative for Backpropagation in Large-Scale Multilayer Neural Networks", The Proceedings of ADVCOMP 2014, Rome, Italy, August 2014, in press.

[5] M. T. Jones, AI Application Programming, 2nd ed., Charles River, 2005, pp. 165-204.

[6] Matlab, The MathWorks, Inc. <http://www.mathworks.com/> [retrieved: August 23, 2014].

[7] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 3nd ed., Prentice Hall, 2009, pp. 727-736.

[8] B. Widrow and M. A. Lehr, "30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation", The Proceedings of IEEE, vol. 78, no. 9, 1990, pp. 1415-1442.

[9] B. J. Wythoff, "Backpropagation Neural Networks: A Tutorial", Chemometrics and Intelligent Laboratory Systems, vol. 18, no. 2, 1993, pp. 115-155.

[10] Y. Yam, "Accelerated Training Algorithm for Feedforward Neural Networks Based on Least Squares Method", Neural Processing Letters, vol. 2, no. 4, 1995, pp. 20-25.

 ISBN: 978-1-61208-367-4

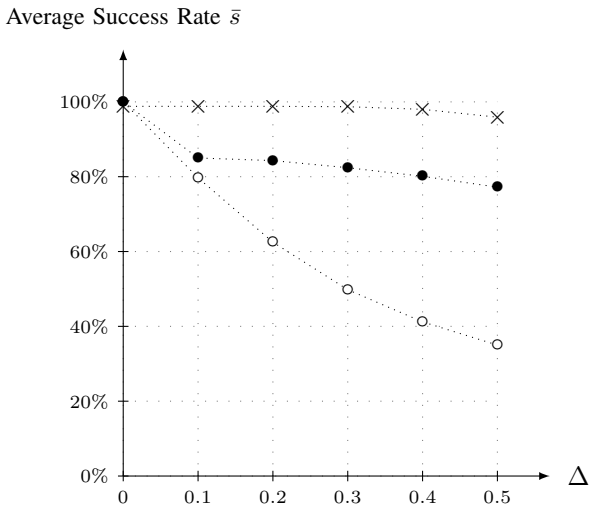


Figure 4. Backpropagation (×), analytic method (○), versus new method using diagonal reinforcement (●), with $M = 10$ (input nodes), $N = 25$ (training points), $H = 24$ (hidden nodes), and $K = 10$ (output nodes).

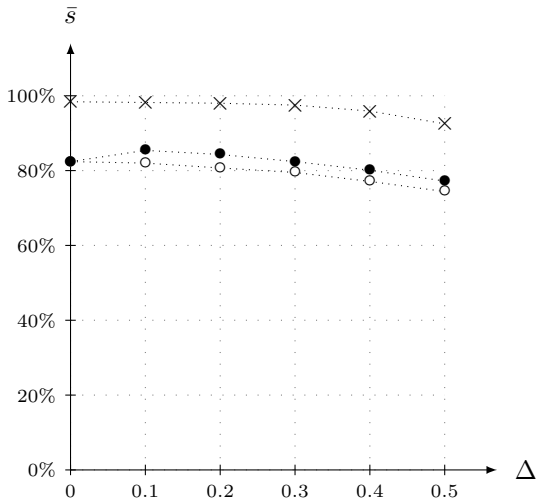


Figure 7. $M = 10$, $N = 25$, $H = 10$, and $K = 10$.

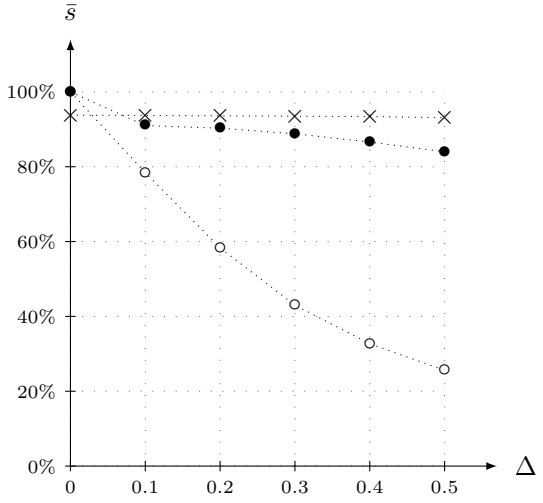


Figure 5. $M = 20$, $N = 50$, $H = 49$, and $K = 20$.

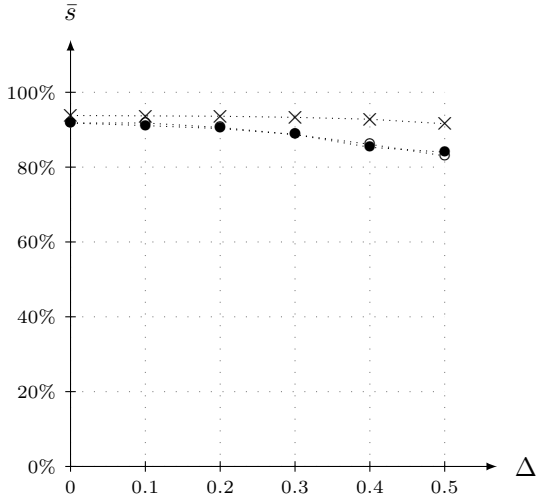


Figure 8. $M = 20$, $N = 50$, $H = 20$, and $K = 20$.

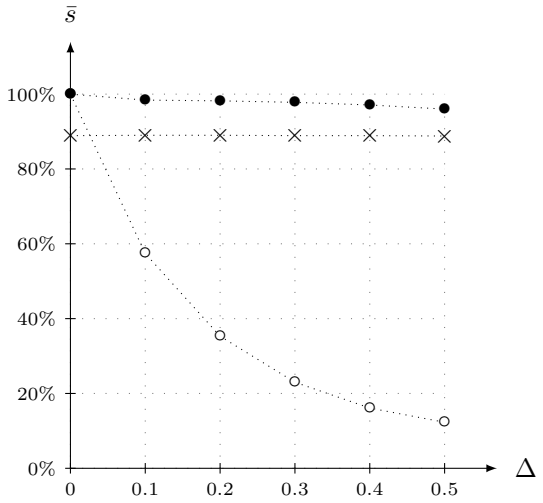


Figure 6. $M = 40$, $N = 100$, $H = 99$, and $K = 40$.

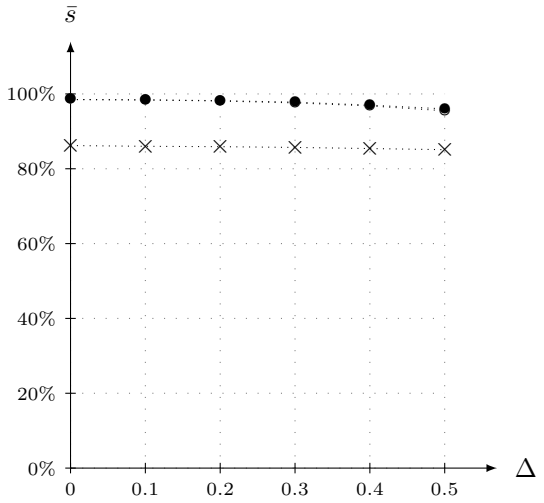


Figure 9. $M = 40$, $N = 100$, $H = 40$, and $K = 40$.

 ISBN: 978-1-61208-367-4

Reorganizing an Offshore Software Project With the Goal of Favoring Knowledge Transfer

Carlo Consoli
IBM
Roma, Italy
carlo.consoli@it.ibm.com

Paolo Rocchi
Università LUISS
Roma, Italy
procchi@luiss.it

Paolo Spagnoletti
Università LUISS
Roma, Italy
pspagnoletti@luiss.it

Pietro Nico
IBM
Roma, Italy
pietro_nico@it.ibm.com

***Abstract*—This paper addresses the management of a software project developed by two groups of professionals, one working locally and the other one working off-shore. After the startup period lasting nearly a year, the project leaders observed that the quality and quantity of the software modules produced by the two teams were not up to expectations while costs had grown up. The project leaders analyzed the types and the amount of software tests required to ensure the quality of the software product. Finally, the management found out that the knowledge transfer process was the real root-causes of the project downfall. The leaders established a new organization as the solution to this problem. They replaced the two large teams with small groups to make communication and cooperation amongst people easier. Immediate evidence has demonstrated the effectiveness of this arrangement.**

Keywords- large software project; project management; offshoring.

I. INTRODUCTION

The basic aspect of offshoring is the notion that some jobs are movable [1]. It may be said that movable jobs are those with little face-to-face customer contact and with high information content. In relation to customer contact, Blinder and others use the term "personally" delivered or "personal" services to describe tasks that require customer contact or physical presence and "impersonal" services to describe tasks that have neither of these prerequisites [2]. In terms of high information content, considerable attention is paid to jobs based on Internet connections, which have greatly reduced the transportation costs of information [3].

The concept of offshoring started in the late 1980s when technology firms discovered emerging countries as basins of untapped resources of high-tech professionals at substantially lower labour costs [4]. Technology firms and Information Technology (IT) departments began to create Offshore Centers of Excellence (OCEs), which, in their early beginnings, related to assisting IT customers and later were devoted to more complex jobs, such as software development and maintenance.

OCEs, in many cases, evolved from a tactical to a strategic role [5]. As OCEs matured in the strategic role, they provided a higher degree of business value, the end goal being to operate in a seamlessly integrated model with the parent organization. However, the model of distributed software development sometime has become a critical success factor in the present global economy [6].

This paper focuses on a crucial aspect of offshore IT projects: the Knowledge Transfer Process (KTP). Issues related to KTPs frequently emerge within the context of IT outsourcing environments and several empirical researches have examined how the development knowledge needs to be shared among technicians and customers, and the quality of the exchanged information must be assured [7] [8] [9] [10]. Many focus on the customers viewpoint, instead [11] approaches the issues from the typical perspective of an offshore software supplier. Lee and others conduct a survey which illuminates the cultural differences which affect the performances of joined Western and Asian software development teams [12].

The present case study deals with this kind of cultural discrepancies and begins with an overview of the software project. Then, it illustrates how the issues rose and finally we shall explain the solution and its validation in relation to human communication and cooperation.

II. SOFTWARE PROJECT PLAN AND ORGANIZATION

An outsourcing project was undertaken by IBM (herein called the "IT Provider") to develop a ticketing application for an Italian company which transports goods and passengers (herein called "XYZ" or the "IT Client"). The aim of the project was to redefine the entire ticket trading system of XYZ, based on on-line sales and ticket offices deployed throughout the Italian territory.

The project began in late 2009 and its high degree of complexity required setting up a specific organizational model for the IT Provider which was deeply integrated with

 ISBN: 978-1-61208-367-4

the structures of the IT Client. This arrangement ensured that the customer and the software producer can cooperate in reaching common goals. Brief profiles of the IT provider entities are given below (see Figure 1.).

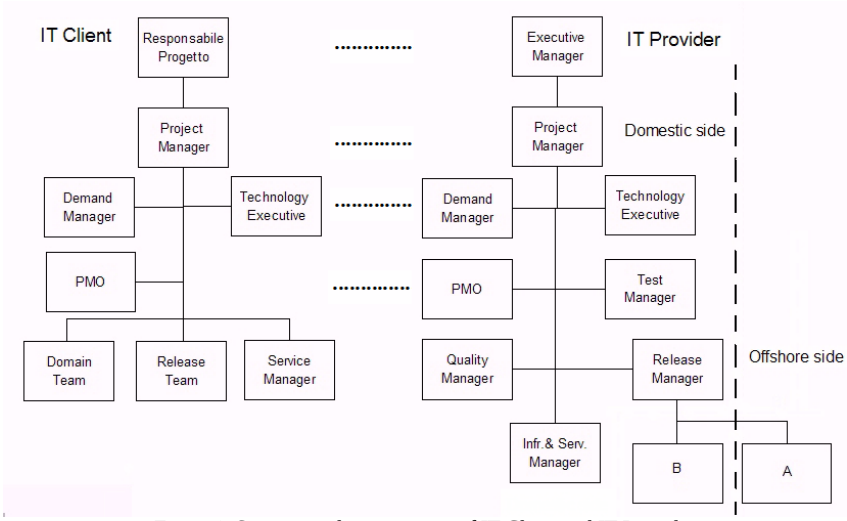


Figure 1. Symmetrical organization of IT Client and IT Provider

- The *Executive Manager* was formally responsible for all the aspects of the work and for the relations with the customer.
- The *Project Manager* was responsible for technical questions.
- The *Technology Executive* was responsible for the overall architectural design of the solution.
- The *Demand Manager* defined the detailed requirements and ensured the adherence of the solution to the customer's needs.
- The *Test Manager* was bound to the level of service.
- The *Project Management Office* (PMO) defined and maintained standards for project management within the organization.
- The *Quality Manager* was responsible for quality assurance.
- The *Release Manager* was responsible for the development and release of the software application, in charge of two different software development teams, one in Italy and one in India.
- The *Infrastructure & Service Manager* ensured that the infrastructure services met requirements in terms of size, performance and availability of the system.

We shall call one or more of the above managers "project leaders". Operations were carried out by the following groups of software practitioners:

(i) The *Release Team* brought out the various modules of the software application supplied to XYZ. This entity registered all the functions implemented and tested in the *Release Note*, a centralized platform used for software management. The Release Note was recognized as the official data handler and on request provided and still provides statistics on the work in progress.

(ii) The *Test Team* included from five to seven testers. The role of this team was to develop and execute test cases, find defects and set the defect status on the Release Note.

(iii) The *Development Team* was subdivided into the *offshore team* A and *domestic team* B.

The former included a variable number of programmers living in India: from 50 to 70, depending on the work load. Five team leaders from Italy managed team A and operated as the front end of the Italian development team B, which included 20 developers primarily involved in the analysis phase. The Indian team was chiefly in charge of coding and had a low level of responsibility.

Teams (i) and (iii) reported to the Release Manager; team (ii) reported to the Test Manager. During the year 2010 teams (i), (ii) and (iii) did not produce significant outcomes. It may be said that 2010 was a break-in period. In the early 2011 practitioners began to work steadily and the project leaders perceived significant difficulties. The quality and quantity of the software modules were not up to expectations and delivery times delayed. The project leaders established that the software modules should be tested in an accurate manner in order to check this unexpected downfall. In the mind of the managers, this control was even supposed to explain the low performances of teams A and B. Testing was executed in Italy, under the direct control of top leaders who surveyed the amount and kind of undertaken tests. They overlooked the number and typology of defects which are more telling on the technical plane, while the amount and kind of tests are appropriate for management purposes.

 ISBN: 978-1-61208-367-4

TABLE I. AMOUNT OF TESTS IN THE FIRST SEMESTER OF 2011

	B	**F**	**S**	**R**	**Total**
1 - Base Functions	73	63	0	0	4,599
2 - Billing Functions	16	6	5	63	411
3 - Advanced Tickets Purchase Options I	46	16	15	69	1,771
4 - Advanced Ticket Purchase Options II	72	29	24	85	4,128
5 – Ticket Purchase with Subscription	44	22	15	114	2,678
6 - Advanced Ticket Purchase Options III	48	90	16	136	6,496
				Grand Total	**20,083**

III. LARGE-SCALE TESTING

The *modules* are the basic components of the ticketing application; an executable version of the module is called a *build* (**B**). Specialists conducted two principal types of software test:

- They undertook **F** *functional tests* on each build. Functional testing focuses on recently implemented functions and overlooks previously implemented functions of the module.
- They executed **R** *regression tests* to validate the overall build, including new and old functions.

The Test Manager arranged **S** *sessions* per module to carry out the regression tests. More precisely, every module was submitted to a series of tests according to the following equations:

NFT (number of functional tests per module) = **B** × **F**
NRT (numbers of regression tests per module) = **S** × **R**
Total (total number of tests per module) =
= NFT + NRT = (B × F) + (S × R)

Table I exhibits data collected in the first semester of 2011. For instance, module # 6 – performing advanced functions in selling tickets – had 48 executable builds each of which underwent 90 functional tests. The manager arranged 16 sessions for module # 6 each of which included 136 regression tests. Thus, module # 6 had 4,320 (=48×90) functional tests and 2,176 (=16×136) regression tests; module # 6 had 6,496 (=4,320+2,176) tests in all.

The noteworthy values of NFT and NRT were basically determined by the amount of detected errors and strengthened the idea that something resulted in the low performances of the offshore and domestic teams. They settled to investigate this failure case by means of further inquiry

The states of the software defects were classified according to the rather usual triage as follows (Table II):

TABLE II. CLASSIFICATION OF SOFTWARE DEFECTS UNDER TESTIONG SESSIONS

New	Defect newly found by the Test Team.
In Progress	Defect subject to ongoing remedial work by the Development Team.
Pending	Defect pending remedial action while the Development Team gathers additional information.
Resolved	Defect remedied by the Development Team.
Reopened	Defect retested by the Test Team and found not to be remedied.
Closed	Defect tested by the Test Team and found to be remedied.

Defects classified as *In Progress*, *Pending* and *Reopened* will generically be termed *Open defects* hereafter. The Test Manager separately surveyed the new and open defects during the first semester of 2011. He noted that new defects were decreasing while the open defects were growing steadily from 50 (February 2011) to 130 (June 2011). This contrasting trend demonstrated that several software errors were causing cascade failures. The Test Manager meant to explore this negative phenomenon using the Release Note that is a software tool for monitoring the status of defects.

The Release Note provided a diagram that exhibits the six states listed in Table 2 (Figure 2); in addition the special block 'Release Note' indicates the status of defects just resolved and under registration by means of the tool Release Note. Teams (i), (ii) and (iii) responsible for handling precise states appear on the far left of Figure 2. For instance, the Development Team was in charge of the defects in the states: Pending, Resolved and In Progress. The flow diagram also shows the transitions of defects from one state to another with the transition frequencies. For instance, 2% of new defects evolve toward the Pending status.

The regular steps to handle a new defect are the following: New → Resolved → Release Note → Closed. But only 88% of new defects went to the status Resolved; 89% passed from Resolved to Release Note; and 84% of defects officially registered were closed. We obtain that a little more than half of the new defects were closed throughout the regular procedure

$$(0.88 \times 0.89 \times 0.84) \approx 0.65$$

In addition, note how 11% of resolved defects could not be registered by Release Notes (100% – 89% = 11%) due to various reasons. As many as 16% of resolved defects (see Release Note → Reopened), and 6% of closed defects (see Close → Reopened) were tested anew for partial corrections. A non-negligible amount of defects crossed the states In Progress → Pending; others followed the pathway: Pending → Resolved → In Progress. Essentially, the flow chart

 ISBN: 978-1-61208-367-4

provided details about the abnormal software production of defects and about the bad handling of those defects carried out by the Teams A and B.

The project leaders calculated the stability of the states in order to better understand the operations achieved by A and B. They counted the number of defects that remained in the same state from approximately February to May 2011 using a simulation program, and obtained the following results:

1) *New*: 4% of the new defects remained in the New state.
2) *In Progress*: 55% of the defects within this state remained so.
3) *Pending*: 92% of the defects within this state remained so.
4) *Resolved*: 11% of the defects within this state remained so.
5) *Reopened*: 1% of the defects within this state remained so.

It is worth explaining how these values – in particular 2) and 3) – do not derive from the priority of defects. Software defects with different urgency levels shared the same destiny. For example, a software error with high priority was revamped and closed in a short while; but a subsequent regression test often placed it into the Open status anew. This cyclic mechanism occurred more than once.

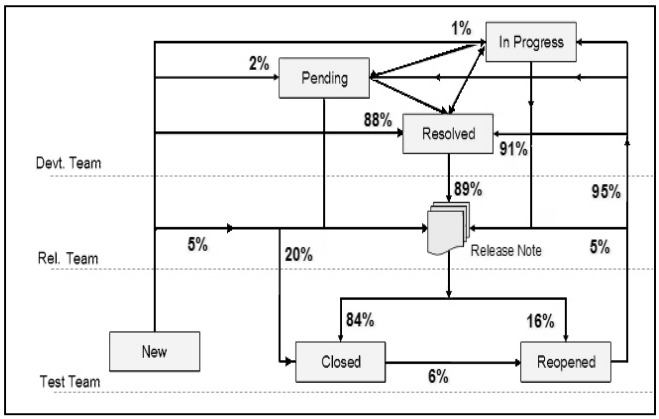


Figure 2. Flowchart of defects' states

Besides the numbers reported above, two practical observations on everyday job clarified the dimension of the project fail. Firstly, groups A and B were overloaded and spent most energy fixing the software defects rather than developing new code. Secondly, the high number of tests incurred time delays and high costs.

In conclusion, quantitative and qualitative data showed how the teams A and B turned out to be ineffective, in contrast with the high professionalism of individuals belonging to the two teams. The project leaders suspected that inefficient KTPs were heavily influencing the operations.

IV. ANALYSIS OF THE SITUATION

A special control group of experts searched for the reasons of this situation and discovered that all the root-causes were related to KTPs in a way. Communication and comprehension between members of the onshore and offshore teams were largely ineffective. In particular, the analytical report of the control group emphasized the following aspects:

I. The teams A and B had been arranged in two very different manners:

a) The Indian team was very large (50 to 70 programmers as described above) and rigidly structured according to hierarchical levels. There were *managers*, *general coordinators*, *area coordinators*, *specialized developers* and *generic developers*. They adopted standard methodologies; they used advanced software tools such as Rational but they followed somewhat rigid work-procedures.

b) Most of the team A members were young and lacking professional experience in large software projects. By contrast, the Italian team B included architects, analysts and developers with extensive experience and knowledge of the target market. The latter group took several details for granted, whereas the former group was completely unaware of technical requirements, the needs of the customer, the defects to correct, etc.

II. Testing was centralized in order to ensure full control of the software development. As a result, the Italian Test Team suffered an overload of activity which stressed the communication between domestic and offshore developers.

III. For team B, it was not a straightforward task to explain the requirements of XYZ and the Indian team. The latter had linguistic difficulties in reading some expressions typical of the Italian transport sector. There were considerable flaws in relation to the delivery of knowledge and knowledge acquisition by the Indians. The offshore team had very little domain knowledge and no understanding of how their development work fitted with the operations of XYZ.

IV. The coding activity of team A was managed by chiefs of the parent organization who viewed the offshore support merely as a low-cost production facility with an abundant supply of cost-effective labor for low-level activities.

Points I, II, III and IV taught the project leaders that difficulties could not be solved through limited counter measures. They decided to rearrange the structure of the entities involved in the project; in particular, they meant to improve the collaboration between teams A and B.

 ISBN: 978-1-61208-367-4

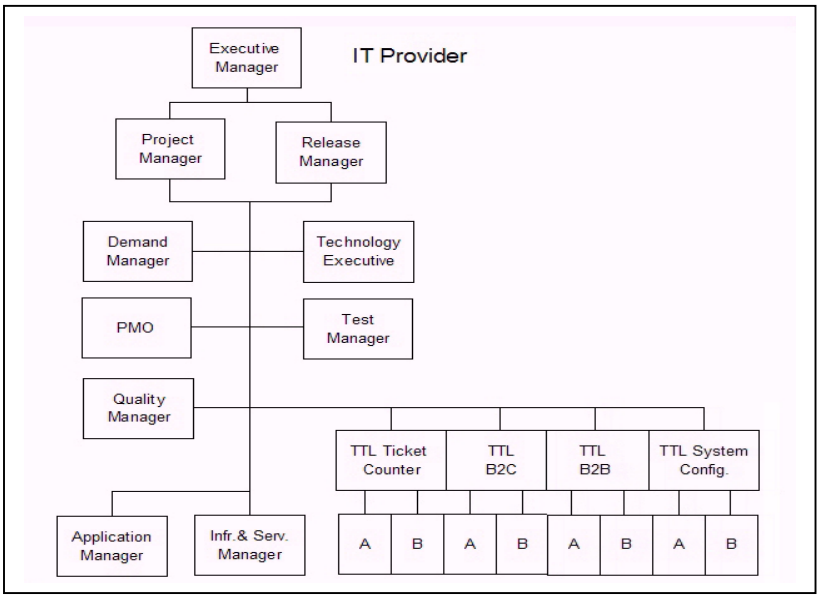


Figure 3. Renewed organization of IT Provider

V. WORKFORCE REORGANIZATION

The principal organizational changes are described as follow.

The domestic and offshore developers are subdivided into eight groups. That is to say, the ex-members of teams A and B comprise eight groups that are paired off and report to four Technical Team Chiefs (TTCs) (Figure 3). Each pair specializes in implementing a precise area of the ticketing application. The areas of railway ticketing are as follows:

- *Ticket Counter* = set of functions related to ticketing
- *Business to Consumer* (B2C) = set of Internet transactions that occur between the transport company XYZ and its customers
- *Business to Business* (B2B) = set of Internet transactions that occur between XYZ and other companies
- *System Configuration* = miscellany of technical functions

Italian and Indian developers become more tightly integrated as they have common and precise goals inside each group. In addition:

- Two Indian experts work in Italy to facilitate integration between multicultural and multilingual groups. This couple of people acquires knowledge of the needs and the characteristics of the Italian market through special training. They are wholly involved in optimizing communication between the domestic and offshore resources.

- The offshore team is assigned to carry out unit tests and functional tests in advance of the corresponding Italian team. A dozen Indian developers learn the ticketing methods of XYZ and are able to suggest corrections for the software modules in case of errors

- The entire testing process is monitored in a "*war room*" which includes experts from the onshore and the offshore side alike. The war room members monitor the status of a module, and analyze and evaluate issues in real time.

- The project leaders simplify the management of the defect states. Abnormal transitions are formally forbidden and, as a result, a negligible number of defects go into the Pending status.

Finally, the Release Manager relinquishes responsibility for software development and assists the Project Manager in gaining a better understanding of the progress of the overall software development.

VI. VALIDATION

In advance of the reorganization, a set of 18 principal functions were identified and scheduled by some project leaders who in addition calculated the resources required for testing these principal functions. Under the original scheme, each function should have required 406 tests of validation; this number includes all kinds of testing, from unit tests to regression tests. The test workload should have required 76

 ISBN: 978-1-61208-367-4

MDs (man-days) and should have caused 15 days of delay (Table III, upper row).

TABLE III. PROSPECTED AND DEFINITIVE AMOUNT OF TESTS REQUIRED BY 18 SOFTWARE FUNCTIONS

	Test per Function	*Total Test Number*	*Total MDs*	*Elapsed Time (days)*
Prospects	406	7308	76	15
Definitive Data	96	1728	18	4

Once the reorganization is completed, the 18 principal functions of the ticketing software applications are tested; the grand total of tests drops down from 7,308 to 1,728; the MDs comes down from 76 to 18 and the elapsed time falls from 15 days to four. This means that the new work organization carries out better software modules and in turn the number of tests necessary to ensure the quality of production decreases. As an example, the regression errors cease to exist. The human resources and the release times dropped down by up to one fourth of the resources previously supplied. A sound net 76% saving on costs and efforts was achieved.

VII. DISCUSSION AND CONCLUSION

This paper is intended to discuss a case of software development which was influenced by KTPs between domestic and offshore teams. The lessons learned by the project managers fit with some modern researches in the sense that KTPs can cause low performance, deprecable quality of software products, time delays and other noteworthy difficulties, while a unified and integrated solution that ensures perfect KTPs does not exist in literature.

It is worth noting that in the beginnings the teams A and B were classified as centers of excellence including skilled professionals. The situation was perfect on the surface and evident obstacles did not emerge in the first year of work. When problems cropped up, the project managers spent some time to discover the root-causes of the problems. Finally, the managers recognized that the cultural gap between the Italian and Indian developers and the diverging daily methods of work were the real origins of the economic losses and inefficient outcomes.

The present paper shows how the project managers have defined a novel governance structure to enable knowledge sharing across organizational boundaries of the off-shore environment. The new teams A and B are subdivided into four sub-teams and are guided by four specialized chiefs who ensure close communication amongst local and remote practitioners. People working in small groups can learn from each other about what is working better; they can get to know each other, keep discussion manageable and allow each discussion to happen in time. In substance, the introduction of small sub-teams turns out to be the organization key measure to enhance KTPs.

REFERENCES

[1] Prasanna B., Tambe Lorin M. Hitt - How Offshoring Affects IT Workers - *Comm. of the ACM* , 53(10), (2010), Pages 62-70.

[2] Bhagwati J.N., Blinder A.S., Friedman B.M. (eds) - Offshoring of American Jobs: What Response from U.S. Economic Policy? - *Proc. Alvin Hansen Symposium on Public Policy*, MIT Press (2009).

[3] Dossani R., Denny N. - The Internet's Role in Offshored Services: A Case Study of India - *Transactions on Internet Technology, Special Issue on the Internet and Outsourcing*, 7(3), (2007), Article No. 15.

[4] Lacity M.C., Willcocks L. - *Global Information Technology Outsourcing: In Search of Business Advantage* - John Wiley & Sons (2000).

[5] Lacity M.C., Khan S.A.,Willcocks L.P. - A Review of the IT Outsourcing Literature: Insights for Practice - *The Journal of Strategic Information Systems*, 18(3) (2009), Pages 130-146.

[6] Prikladnicki R., Audy J.L.N. - Process Models in the Practice of Distributed Software Development: A Systematic Review of the Literature - *Information and Software Technology*, 52(8), (2010), Pages 779-791.

[7] Mohamed A., Arshad N.H., Abdullah N.A.S. - Influencing Factors of Knowledge Transfer in IT Outsourcing - *Proc. 10th WSEAS Intl. Conf. on Mathematics and Computers in Business and Economics*, (2009), Pages 165-170.

[8] Faiz, M.F., Qadri, U., Ayyubi, S.R. - Offshore Software Development Models - *Proc. Intl Conf on Information and Emerging Technologies*, (2007), Pages 1-6.

[9] Prabhu N.V.A., Latha R., Sankaran K., Kannabiran G. - Impact of Knowledge Management on Offshore Software Development: An Exploratory Study - *Proc. Third Intl. Conf. on Advanced Computing*, (2011), Pages 121-128.

[10] Pilatti L., Audy J.L.N. - Global Software Development Offshore Insourcing Organizations Characteristics: Lessons Learned from a Case Study - *Proc. Intl. Conf. on Global Software Engineering*, (2006), Pages 249-250.

[11] Rajkumar T.M., Mani R.V.S. - Offshore Software Development: The View from Indian Suppliers - *Information Systems Management*, 18(2), (2001), Pages 63-73.

[12] Lee D., Smith A., Mortimer M. - Cultural differences affecting quality and productivity in Western/Asian offshore software development - *Proc. of the 3rd International Conference on Human Computer Interaction*, (2011), Pages 29-39.

 ISBN: 978-1-61208-367-4

Challenges of the Existing Tools Used in Global Software Development Projects

Mahmood Niazi, Sajjad Mahmood, Mohammad Alshayeb, Ayman Hroub

Department of Information and Computer Science, King Fahd University of Petroleum and Minerals, Saudi Arabia

{mkniazi, smahmood, alshayeb, aymanh}@kfupm.edu.sa

***Abstract*— Global Software Development (GSD) has been embraced by organizations due to the availability of highly trained software engineers at a relatively low cost. GSD is not a risk free activity as several GSD failures have been reported. It is anticipated that the appropriate use of available software tools can play an important role in overcoming some of the risks associated with management of GSD projects. However, there are many challenges in adopting the existing tools in globally distributed projects. The objective of this paper is to identify challenges of existing tools used in GSD projects. We have used a Systematic Literature Review (SLR) approach by applying customized search string derived from our research questions. We have identified 105 papers that discuss the challenges of the existing GSD tools. We have identified key challenges for adopting existing tools in GSD projects, such as: "difficulties in adopting and learning existing tools", "inappropriate use of tools", "lack of coverage of GSD processes" and "lack of security and privacy". Based on our SLR results, we suggest that GSD organizations should address these challenges in order to compete in the GSD business.**

Keywords-Global software development; Challenges; Systematic Literature Review; Software Tools.

I. INTRODUCTION

Global software development (GSD) is becoming a promising methodology to build quality software at a low development cost and short time-to-market. GSD is the process where a company either has its software developed by geographically distributed teams or contracts all or parts of its software development activities in return for remuneration [1] [2]. A number of software organizations in the developed world have outsourced their software development projects to emerging countries (e.g., China and India) where they have access to large pools of highly trained software engineers at relatively low cost.

The adoption of GSD has introduced potential benefits as well as challenges for software organizations. GSD has dramatically changed the business economics in the overall software industry by, for example, utilizing time-zone differences to organize round-the-clock project development life cycle. Furthermore, globalization of software projects allows companies to employ software engineers with required skills to work on a project from different geographical locations. On the other hand, the geographically distances and cultural differences between globally distributed teams have also introduced new challenges, such as: difficulty in maintaining collaboration, coordination and communication [3] [4] [5] [6].

Lately, researchers [7] [8] have indicated that readily available software tools can help in overcoming challenges associated with development and management of software projects by GSD teams. Document management systems, wiki and blog features of software tools have been used for knowledge management among GSD teams. Similarly, social computing tools, such as: Skype, Twitter, etc., are being used in multi-site GSD projects to provide additional communication channels. This not only enables real-time communication but also allows knowledge sharing and instant feedback from different teams involved in the project [9].

Despite the increased use of software tools in GSD projects, little research has been carried out to comprehend the challenges associated with the adaptation of existing tools in the GSD environment. We also need to investigate how to help organizations in selecting suitable tools to ensure the successful outcome of projects and to maintain long lasting relationships between the clients and the vendors.

In this paper, we aim to identify challenges, via systematic literature review, of using existing software tools in GSD projects. Identifying these challenges will assist GSD organizations in better development and management of GSD projects. Our long-term research goal is to develop a global project management readiness framework to assist software development organizations in measuring and improving their project management readiness prior to starting global activities. To achieve this, we intend to address the following research question in this paper:

RQ: What are the challenges of existing tools used in GSD projects?

The rest of this paper is organized as follows: Section II provides the GSD background. Section III describes the research methodology. In Section IV, we present the initial results with analysis and we conclude in Section V.

II. BACKGROUND

GSD is a software engineering paradigm aiming at developing high-quality software in low-wage countries at reduced cost [1]. The various types of GSD projects can be grouped into two categories, namely, outsourcing on the basis of geographical location and outsourcing on the basis of relationship [10]. On the basis of geographic distance between vendors and clients, outsourcing is further

 ISBN: 978-1-61208-367-4

categorized into three types: onshore outsourcing, near-shore outsourcing and offshore outsourcing [11]. Onshore outsourcing is also called domestic outsourcing, which consists of both domestic vendors and domestic clients [12]. This means that both (vendor and client) organizations are located in the same country. Near-shore outsourcing or simply near shoring is defined as the transfer of software development work to a nearby foreign country to reap lower labor cost advantages [13]. Offshore software development outsourcing refers to outsourcing in a geographically distant country. The offshore activities have been going on since the last decade and are growing rapidly [14]. The major vendor countries for offshore outsourcing are India, Ireland, China and Russia whereas the client countries are USA, UK, Australia and Japan [15].

There are many reasons for initiating GSD project [13] [14] [15] [16]. Client organizations benefit from GSD because vendors in developing countries (offshore vendors) typically cost one-third less than onshore vendors and even less when compared with in-house operations [17]. Among many other reasons for GSD, generally client organizations outsource their software development work to offshore locations to gain quality advantages, improve their skills, access to leading-edge technologies and focus on their core competencies [13]. Conversely, there are many risks in the GSD, such as: temporal incompatibility, cultural differences and hidden costs [18] [19]. IT Week magazine reported that eight out of every ten firms that outsourced their software development project to an offshore vendor faced major problems due to insufficient preparation and poor management by both client and vendor organizations [20].

There are many reasons for these problems. One of the major issues is the lack of awareness about software tools support for GSD projects and what features they provide to support globally distributed software development projects. Understanding issues related to adoption of suitable software tools to support different phases of GSD can help in achieving greater success in GSD projects. In this paper, we conduct a systematic literature review to identify challenges of the existing tools used in GSD projects.

III. RESEARCH METHODOLOGY

In this study, we followed the systematic literature review (SLR) process to find the data required to address our research question [21]. SLR is a defined process that aims at providing an exhaustive summary of literature by identifying and analyzing published studies relevant to the investigated research question [22]. SLR may use scoring of the levels of evidence or statistical techniques (meta-analysis) to combine results of the identified studies. Therefore, the results obtained from SLR may provide a better insight than might be in ordinary literature review or surveys.

To conduct the SLR, we developed the systematic review protocol. The protocol describes the plan for the review. The SLR includes the following main steps:

- Define the search strategy
- Search for relevant studies
- Select relevant studies
- Perform study quality assessment
- Extract data from the finally selected studies
- Analyze the extracted data

In order to achieve the objective of this paper, we set the following research question:

RQ1: What are the challenges of existing tools used in GSD projects?

For the above research question, we identified the following major terms:

- POPULATION: GSD projects.
- INTERVENTION: GSD challenges of existing tools.
- OUTCOME OF RELEVANCE: the challenges of existing tools used in GSD projects.

Finally, after a number of trials, we identified the following search string that is used in this study:

{Challenges OR limitations} AND
{Technology OR tools} AND
{GSD OR Global Software Development OR Global Software Engineering OR Global Software Testing OR Software Outsourcing OR Software Offshoring OR Geographically Distributed Software Development}

We used the following digital libraries to run our search string: ACM Digital Library, IEEE Explore, Science Direct, Springer Link and John Wiley.

The following inclusion criteria were used:

- The paper should be related to GSD.
- The paper should clearly mention at least one challenge or limitation either for a certain tool used in GSD or for technology used in GSD in general.
- The following exclusion criteria were used:
- Non-English papers were excluded.
- White papers and technical reports were excluded.
- Papers related to technology used by distributed teams other than software engineering were rejected.

To address our research question, we extracted data from the finally selected papers. The following data were extracted from each paper: publication type, authors, publisher, publication name, publication date, organization size, project size and challenges of tools used in GSD projects.

IV. INITIAL RESULTS AND DISCUSSION

Table I shows the SLR process results. A total of 1318 papers were retrieved after entering the customized search string in the digital databases. After reading the title and the abstract, 318 papers were selected. Finally, 105 papers were selected after reading the whole paper.

The finally selected papers were categorized according to the nine study strategies as shown in Table II. Most of the articles have used case study research method.

Table III provides details of the publication venues for the papers identified in our SLR study. Nineteen papers are from Finland, eighteen papers are from USA, nine papers are from Germany, six papers are from Brazil, five papers are from the Netherlands and four papers are from UK.

 ISBN: 978-1-61208-367-4

TABLE I. SEARCH EXECUTION

Library Name	Total Results	Initial Selection	Final Selection
IEEE Explore	980	250	82
ScienceDirect	88	13	6
ACM	180	34	7
John Wiley	70	21	10
Total	1318	318	105

TABLE II. STUDY STRATEGIES USED

Study Type	Count
Case Studies	29
Literature Review	16
Implementation	21
Interviews	10
Empirical	8
Systematic Literature Reviews	6
Systematic Mapping Review	4
Social Network Analysis	1
Others	10
Total	105

From the accepted papers, 8 challenges of the tools used in GSD projects were extracted as shown in Table IV.

TABLE III. STUDY COUNTRIES

Country	Count	Country	Count
Argentina	2	Malaysia	1
Australia	3	Mexico	3
Brazil	6	Netherlands	5
Canada	3	New Zealand	2
China	1	Norway	2
Denmark	4	Pakistan	1
Finland	19	Spain	3
Germany	9	Sweden	2
Iran	1	Switzerland	1
India	4	UK	4
Ireland	5	USA	18
Italy	4	Venezuela	1
Latvia	1		

In our study, the most common challenge of the tools used in GSD projects is “inappropriate use of synchronous and asynchronous communication tools” (43%) as shown in Table IV. This can be due to multiple reasons, such as:

- The synchronous tools are useless when the time difference among the remote teams is more than 8 hours, i.e., no working time overlap between different teams and thus they cannot utilize these technologies unless one team shifts the working hours.
- The synchronous interaction causes interruptions for the employees in their daily work as often unnecessary communication is performed. Some team members may nudge each other without knowing the status of the receiver (i.e., if receiver is busy in an important meeting or meeting a crucial deadline).
- During the use of synchronous and asynchronous communication tools, the GSD professionals often face problems due to cultural and language differences.
- Asynchronous communication tools like email or forums are not appropriate for solving conflicts and technical interactions due to their late responses.

TABLE IV. LIST OF CHALLENGES

Challenges	Freq. (n=105)	%
Inappropriate use of synchronous and asynchronous communication tools	45	43
Difficulties in adopting and learning existing tools for GSD projects	31	30
Lack of coverage of GSD development processes.	26	25
Lack of data integration due to different collaboration tools used in GSD projects.	21	20
Lack of support for collaboration and group decision making.	12	11
Lack of security and privacy in communication and collaboration tools.	10	10
Lack of awareness of existing tools used in GSD projects.	6	6
Lack of ability to track the progress of tasks assigned to team members in GSD projects.	5	5

The second frequently mentioned challenge is “adopting and learning new tools for GSD projects” (30%). This is because there is an increasing pool of software tools that can be used in GSD projects and selecting and adopting the most appropriate tool from this pool is a challenging task. This may be due to the absence of a well-defined procedure to select the best tool. In addition to that, some people are reluctant to use some tools due to cultural issues. Moreover, some people are resistant to change and therefore they do not like to replace their current tools with new ones.

All these problems show that most of the GSD organizations do not design the adequate communication strategies. It is very important to early develop a good communication strategy in order to reduce misunderstandings between stakeholders from different country cultures [2].

The other highly mentioned challenge is the “lack of coverage of GSD processes” (25%). None of the existing tools cover all processes in the GSD life cycle. Most of the tools are dedicated for a specific function like communication, testing, requirements engineering etc. As a result, the GSD companies need to have many different tools

 ISBN: 978-1-61208-367-4

to perform different GSD functions. In addition, there are important GSD areas, which are rarely covered by the existing tools, such as: risk management, requirements engineering, issue tracking and social awareness.

20% of the articles mentioned “lack of data integration due to different collaboration tools used in GSD projects”. This incompatibility is due to the absence of standards for the different tools vendors. This challenge complicates the data transfer among these different tools.

10% of the articles have mentioned “lack of security and privacy in communication and collaboration tools” as a current technology challenge in GSD projects. These security issues are related to source code and project information that is exchanged over the web. There are also privacy concerns for the team members when using these tools for informal communication, such as: instant messaging or social media.

V. LIMITATIONS

We limited our SLR study to four research publication databases. However, there are other related research databases which we did not consider in our study, which may have relevant publications. Furthermore, with the increasing number of research papers published on this topic, some recent and relevant publications could have been missed at the time of consolidating the results of the SLR. Nevertheless, we believe that our presented results are comprehensive and cover the most relevant published literature.

VI. CONCLUSION AND FUTURE WORK

There is a growing interest in GSD for software development companies. In addition to the challenges that are related to the GSD business nature and cultural differences, there are other challenges associated with the tools used in GSD. In this paper, we identified challenges of the software tools used in GSD projects from the existing literature. We identified 105 papers that discuss the challenges of the existing GSD tools and technologies. These challenges range from unsuitable or missing features in these tools to the non-existence of tools in some GSD areas. There are other challenges related to the cultural and time zones difference issues. In addition, the existing tools are neither comprehensive nor compatible with each other to allow tools integration.

The next step is to conduct an empirical study to support our findings. This includes designing a questionnaire in the light of our findings and gathering information from the software industry professionals about the challenges related to the tools used in GSD projects.

The overarching objective of this research work is to develop a global project management readiness framework to assist software development organizations in measuring and improving their project management readiness prior to starting any global software development activities.

ACKNOWLEDGMENT

The authors would like to acknowledge the support provided by the Deanship of Scientific Research at KFUPM, Saudi Arabia, under Research Grant 11-INF2152-04. We are also thankful to Dr. Narciso Cerpa (University of de Talca, Chile) for reviewing our protocol.

REFERENCES

[1] S. U. Khan, M. Niazi, and R. Ahmad, "Factors influencing clients in the selection of offshore software outsourcing vendors: an exploratory study using a systematic literature review," Journal of Systems and Software, vol. 84, no. 4, (2011), pp. 686-699.

[2] S. U. Khan, M. Niazi, and A. Rashid, "Barriers in the selection of offshore software development outsourcing vendors: an exploratory study using a systematic literature review," Journal of Information and Software Technology, vol. 53, no. 7, (2011), pp. 693-706.

[3] G. Aranda, N., A. Vizcaíno, and M. Piattini, "A framework to improve communication during the requirements elicitation process in GSD projects," Requirements engineering, vol. 15, no. 4, (2010), pp. 397-417.

[4] Benjamin, B. M. Shao, and J. S. David, "The impact of offshore outsourcing on IT workers in developed countries," Communications of the ACM, vol. 50, no. 2, (2007), pp. 89 - 94.

[5] L. I. Charalambos, and N. Robbie, "A risk profile of offshore-outsourced development projects," Communications of the ACM, vol. 51, no. 6, (2008), pp. 89-94.

[6] H. Christiansen, Munkebo, "Meeting the challenge of communication in offshore software development," Software Engineering Approaches for Offshore and Outsourced Development. Lecture Notes in Computer Science, vol. 4716, no. (2007), pp. 19-26.

[7] J. Portillo-Rodríguez, A. Vizcaíno, M. Piattini, and S. Beecham, "Tools used in Global Software Engineering: A systematic mapping review," Information and Software Technology, vol. 54, no. 7, (2012), pp. 663-685.

[8] M. A. Storey, C. Treude, D. Van, A., and L. T. Cheng, "The Impact of Social Media on Software Engineering Practices and Tools," Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research, (2010), pp. 359-364.

[9] M. Niazi, S. Mahmood, M. Alshayeb, A. Baqais, and A. Q. Gill, "Motivators of Adopting Social Computing in Global Software Development: Initial Results," World Congress on Engineering 2013 (WCE 2013), London July 2013, ISBN: 978-988-19251-0-7, (2013), pp. 409-413

[10] S. U. Khan, Software outsourcing vendors' readiness model (SOVRM), PhD thesis, Keele University, UK (2011).

[11] P. A. Laplante, T. Costello, P. Singh, S. Bindiganavile, and M. Landon, " The who, what, why, where, and when of IT outsourcing," IEEE IT Professional, vol. 6, no. 1, (2004), pp. 19 - 23.

[12] B. Shao, David, J.S., "The impact of offshore outsourcing on IT workers in developed countries.," Communications of the ACM, vol. 50, no. 2, (2007), pp. 89 - 94.

 ISBN: 978-1-61208-367-4

[13] A. Stetten, v., D. Beimborn, E. Kuznetsova, and B. Moos, "The Impact of Cultural Differences on IT Nearshoring Risks from a German Perspective," in Proceedings of the 43rd IEEE Hawaii International Conference on System Sciences, (2010), pp. 1-10.

[14] D. Smite, C. Wohlin, T. Gorscheck, and R. Feldt, "Empirical evidence in global software engineering: a systematic review," Empirical Software Engineering, vol. 15, no. 1, (2010), pp. 91-118.

[15] N. V. Oza, An empirical evaluation of client - vendor relationships in Indian software outsourcing companies, PhD thesis, University of Hertfordshire, UK (2006).

[16] A. A. Bush, A. Tiwana, and H. Tsuji, "An Empirical Investigation of the Drivers of Software Outsourcing Decisions in Japanese Organizations," Information and Software Technology Journal, vol. 50, no. 6, (2008), pp. 499-510.

[17] L. McLaughlin, "An eye on India: Outsourcing debate continues.," IEEE Software, vol. 20, no. 3, (2003), pp. 114-117.

[18] H. Holmstrom, E. O. Conchuir, P. J. Agerfalk, and B. Fitzgerald, "Global Software Development Challenges: A Case Study on Temporal, Geographical and Socio-Cultural Distance," Global Software Engineering, 2006. ICGSE '06. International Conference on, (2006), pp. 3-11.

[19] D. Damian, L. Izquierdo, J. Singer, and I. Kwan, "Awareness in the Wild: Why Communication Breakdowns Occur," Global Software Engineering, 2007. ICGSE 2007. Second IEEE International Conference on, (2007), pp. 81-90.

[20] L. Mary, and R. Joseph, "Effects of offshore outsourcing of information technology work on client project management," Strategic Outsourcing: An International Journal, vol. 2, no. 1, (2009), pp. 4-26.

[21] B. Kitchenham, and C. Charters, Guidelines for performing Systematic Literature Reviews in Software Engineering. Keele University and Durham University Joint Report - EBSE 2007-001, EBSE 2007-001 (2007).

[22] M. Staples, and M. Niazi, "Experiences Using Systematic Review Guidelines," Journal of Systems and Software., vol. 80, no. 9, (2007), pp. 1425-1437.

 ISBN: 978-1-61208-367-4

ARTIST Technical Feasibility Tool: Supporting the Early Technical Feasibility Assessment of Application Cloudifications

An approach for estimating the complexity of a cloudification project in a pre-modernization stage

Juncal Alonso Ibarra, Leire Orue-Echevarria, Zurik Corera Seoane
ICT-European Software Institute Division,
Tecnalia
Zamudio, Spain
juncal.alonso@tecnalia.com, leire.orue-echevarria@tecnalia.com, zurik.corera@tecnalia.com

Jesus Gorroñogoitia, Burak Karaboga
Atos Research & Innovation
Madrid, Spain
jesus.gorronogoitia@atos.net, burak.karaboga@atos.net

***Abstract*— Modernizing an IT system is a long, complex journey. The pre-migration phase is the starting point of each migration project where the decision to transform the legacy rather than to rewrite it has to be taken. In order to support this decision making, the ARTIST European project [1] proposes a technical feasibility analysis to as much technical information as possible about the legacy application itself and about the required technical tasks to migrate its components. This paper presents a technical feasibility analysis which relies on Cloud Migration Point approach to estimate the cost of the migration (in terms of required effort) and incorporates techniques such as Model Driven Reverse Engineering, software complexity metrics or Domain Specific Language-based heuristics to automate this process as much as possible, although leaving to the user the knowledge and control all over the entire process**

Keywords-Software modernization, technical feasibility, software complexity, cloud computing, migration strategy.

I. INTRODUCTION

Prior to facing a challenging project such as a software migration one, which may involve not only changing the way companies will deliver their software but also, probably, their business model and organizational processes, software vendors need to analyse if what they want to achieve, is actually feasible for them in terms of technology, processes and business.

This paper presents an approach for a technical feasibility analysis of a migration of an application to the cloud. The main aim of this analysis is twofold. On one hand, support the establishment of the most suitable migration tasks and on the other hand, provide an estimation of the required effort to implement these migration tasks with the final goal of supporting the decision making process prior to a modernization project.

II. MOTIVATION

Research literature and real industrial migration projects have documented several general procedures to estimate the cost and efforts required by a migration process, and therefore deciding on its feasibility.

Both analogy based estimation [2], that is, by comparing current migration project with other undergone migration projects and estimation given by experts' judgment [3] uses the knowledge in previous similar migration experiences, gained by experts to evaluate and estimate the complexity and efforts to undertake a new migration mission. Unfortunately, these approaches cannot be applied to migration project towards the Cloud, since the Cloud paradigm adoption is relatively recent, whereby the number of documented migration projects of legacy software to the Cloud is scarce [6, 7].

The most popular estimation approach is based on algorithmic models [4] that propose mathematical models to derive a quantitative estimation of migration costs based on identified costs factors. Although this approach also requires historical data in order to evaluate some parameters introduced by the mathematical models (i.e. weights in the model), its applicability is more generic than previous approaches, and therefore more suitable for a wider range of migration projects.

In order to estimate software development costs using metrics for software size measurement, some algorithmic methods based on Function Point Analysis (FPA) [5] have been proposed in literature. The FPA cost estimation is based on the analysis of software requirements.

FPA-based approaches can be more appropriate to estimate the complexity and provide effort/cost estimations (by historical data comparison) of migration tasks. In particular, FPA function points, in the context of a migration to Cloud project, can be mapped into migration tasks [6]. The systematic estimation of efforts required to migrate a legacy application into the Cloud has received less attention in the research community, notably because the migration to Cloud is a relative new concern. Up to our knowledge, only one work has proposed a systematic methodology for effort estimation of Cloud migration projects, namely Cloud Migration Point (CMP) [7], an adaptation of the FPA approach for software size estimation applied to the context of Cloud migration.

Complementing FPA-based approaches, there exist others based on software size estimation, including software complexity estimations. However, these methods can hardly be used on their own when wishing to estimate the size and complexity of the developments required migrating a legacy

 ISBN: 978-1-61208-367-4

application to the Cloud [8], because they do not offer enough information. Nonetheless, software size/complexity estimations on components of existing software systems can be used to classify the complexity of migrations tasks performed on these components, by comparing computed complexity metrics with historical data [9]. In particular, coupling metrics seems to help in the re-factoring of subsystems in an effective way to achieve the lower cost and high re-usability [10], which are factors to take into account when migrating to Cloud.

III. ARTIST APPROACH FOR TECHNICAL FEASIBILITY ASSESSMENT

A. *Mission and scope*

The ARTIST Technical Feasibility Tool (TFT) aims at supporting users on the early technical assessment of the migration of their applications to the Cloud. At this early stage (e.g. pre-migration phase in ARTIST Methodology [11]), the users need support to evaluate the feasibility of the migration, attending its technical aspects, since even for a very simple application, its migration to the Cloud may require non negligible efforts and concrete expertise to be accomplished. Moreover, the support for decision making requires a detailed breakdown of the migration process into a set of technical tasks, not only to estimate their required efforts, but also to identify other resources needed to accomplish every task, including the selection of the appropriate technical expertise or even the detection of dependencies among tasks or other technical intricacies.

B. *Functional description*

TFT works on Model Driven Engineering (MDE) representations (e.g. models) of the applications, particularly UML component models, offering to the users the following features:

- Visualization of components or features of the legacy application and the selection of those to be affected by the migration.
- Visualization of migration goals, which ultimately will drive the migration process. Migration goals can be obtained from the Cloud maturity assessment obtained through the ARTIST Maturity Assessment Tool (MAT) [12] or expressed by the user using the ARTIST Goal Modeling Editor [13].
- Identification of the required migration tasks on affected components. TFT suggests migration tasks per component. TFT allows users to confirm these tasks (optionally, TFT tries to select some tasks by default, but the user is able to override this selection anytime).Selection of weighted complexity estimations for every task type from expert judgment figures, initially taken from [7]. These figures provide task complexity weights estimated by experts based on accumulate experiences.
- Computation of complexity estimations for every component, calculating some metrics, in particular those metrics that estimate their maintainability.
- Computation of complexity estimations for a single task, as a function that considers both the complexity of the component affected by the task and task complexity itself
- Computation of effort estimations for a single task, as proportional to the computed task complexity, where the proportionality weight is given by expert judgment.
- Computation of global migration effort, by summing over individual migration task, for each migrated component.

C. *Technical approach*

Our implementation of TFT extends the CMP approach by automating some steps, using techniques explored by ARTIST such as Model Driven Reverse Engineering (MDRE), Software Metrics or Domain Specific Language (DSL)-based heuristics, notably to extract knowledge of the application, propose migration strategies and estimate the component complexity. CMP based computation of migration efforts is mostly conducted manually. On the contrary TFT is aiming to automate this process as much as possible, although leaving to the user the knowledge and control all over the entire process.

TFT approach to estimate the cost of the migration is based on the analysis of the migration requirements. Therefore, the specification of the overall objectives of the migration, that is, the migration goals, combined with the component-specific migration requirements and the preliminary Cloud target selection are inputs that will drive the TFT analysis. TFT leverages on high level model representations of the application, from which TFT elaborates a detailed breakdown analysis into components or features and creates a detailed structural breakdown of the migration process per legacy component. For such, TFT extracts legacy components from the high level model representations of the application, analyses their relationships and dependencies, determines their type (i.e. data sources, data entities, distributable services, controllers, views, etc.), estimates their complexity and maintainability (and possibly other metrics), and finally reports all these findings to the user in a component inventory view. TFT uses sources of domain-specific information, like expert judgment, to define heuristics that are used to infer the most appropriate migration strategies. These strategies are instantiated as migration tasks, for each component selected for migration, aiming at fulfilling the overall migration goals and the specific component migration requirements, addressing the Cloud target selection as well. TFT encodes these heuristics, used for task suggestion, in rules defined with a concrete domain specific language (DSL), in particular, tha JBoss Drools [14] DSL and engine is used., This approach avoids hardcoding expert judgment on TFT code implementation, which provides greater flexibility to extend the TFT knowledge in the future.

 ISBN: 978-1-61208-367-4

IV. TECHNICAL FEASIBILITY TOOL: DESIGN AND IMPLEMENTATION

A. General architecture

In Fig.1 the general architecture of the Technical feasibility Tool is depicted, and explained in section B.

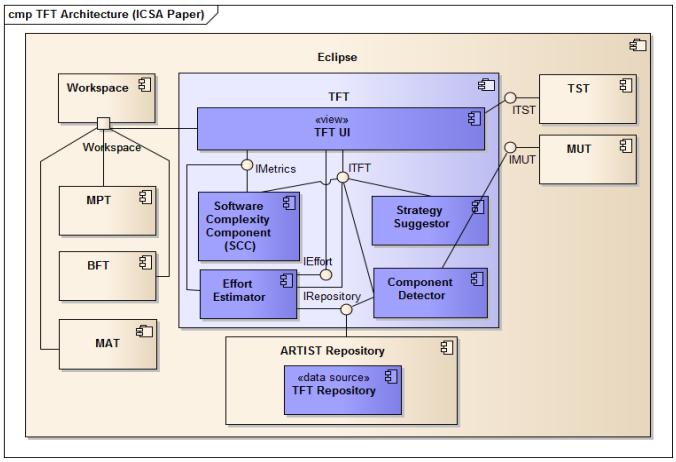


Figure 1. TFT general architecture

B. TFT components in detail

TFT consists of a set of Eclipse views and other widgets and wizards, a set of backend components and a set of external dependencies with other ARTIST components and tools, accessed through well-defined interfaces.

TFT UI complements the ARTIST Eclipse perspective with its collection of views on which the functionality of TFT is offered to the user:

- Navigator view: to browse and select existing legacy application projects
- Modelling view: to browse and annotate platform specific/platform independent models (PSM/PIM) component views provided by the ARTIST Model Understanding Tool (MUT) [15].
- Annotation View, provided by the ARTIST Target Specification Tool (TST) [15], which collects existing migration goals/requirements and provides support to annotate the existing legacy models in order to express additional migration goals.
- Inventory of components View: this TFT view collects the components from the component model and suggests migration strategies for each of them. The estimation of efforts for these migration strategies are also calculated and are shown to the user in a range of low, average and high for each migration strategy. The view allows modifying the migration strategies that affect them from a list of compatible strategies depending on the components' properties. This view also allows the user to select/deselect components to be considered to be migrated or not.
- Migration Goals View: allows user to browse and enable/disable the migration goals provided by MAT.
- Metrics view: this view allows selecting the metrics to be calculated for a selected component and displays the metrics figures.
- Effort estimation report view: this view reports the estimated effort for the overall migration project and individual migration tasks.

The TFT-UI makes use of these views, which are heavily dependent on RCP components such as Standard Widget Toolkit and JFace. Eclipse Workbench components are also used to make contributions to the Eclipse UI itself. TFT contributes to context menus of files with "uml" and "di" extensions and Papyrus [16] containers, with actions to open the Inventory View, and to context menus of files with XML extension to open the Migration Goals View. The TFT plugin also adds a listener to the opened component diagram files which listens the changes done to the file via using EMF/UML2 [17] or Papyrus editors.

TFT relies on several backend components to provide business logic support to TFT-UI.

- Components Detection component**:** It analyzes high level EMF Ecore UML2 PSM/PIM component models of the selected legacy application. The component uses EMF-Query to filter and EMF-Core and UML2 to analyse and modify the input model.
- Software Complexity component which computes a set of metrics on selected components. This component is explained in detail in the next section.
- Migration Strategy Suggestion component: It is responsible for analysing the components of the non-cloud compatible application and the relationships between them and suggesting certain migration strategies for each component to assist the user in the pre-migration process. Strategy suggestion process relies on a set of Drools rule defined in a DSL-based rule language which is interpreted by JBoss Drools. The strategy suggestion process is handled by the rule engine which is implemented using JBoss Drools
- Effort Estimation component: This component estimates the effort required to accomplish each required migration strategy suggested. The effort calculation is based on the migration strategy complexity and the complexity of the affected component(s). Strategy complexity is calculated using historical data and the expert knowledge encoded in the DSL based rules. Component complexity is provided by the Software Complexity Component. The final effort metric values are also based on expert knowledge combined with the complexity metrics.

 ISBN: 978-1-61208-367-4

- TFT Repository**:** This component stores historical data and heuristics required to estimate efforts.

1) Software Complexity Component

In order to evaluate the effort required to perform a migration task, TFT analyses several parameters as explained above in the paper. One of these parameters is the complexity related to the legacy software.

The estimation of the complexity of the legacy software is performed, by the Software Complexity Component (SCC). It provides information about how complex the legacy software is in terms of easiness to evolve it to the Cloud paradigm. This information is provided by means of software complexity metrics.

Software complexity has been defined and calculated in a vast variety of ways in the last years. Upon closer examination, these are some several commonly used metrics:

- McCabe Cyclomatic Complexity (v(G)) [18]
- Weighted Methods per Class (WMC) [19]
- Afferent Coupling (Ca) [20]
- Efferent Coupling (Ce) [20]
- Instability (I= Ce / (Ca + Ce)) [21]
- Number of Interfaces [21]

The correlation of these metrics is of highly importance, as a variation in one of them has an impact on the others. Literature has studied this correlation mainly for maintainability concerns which is defined by IEEE standard glossary of Software Engineering [22] as "the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment".

The Compound MEMOOD method presented in [23], based on the MEMOOD model [24], creates a maintainability model based on the creation of 4 models: 1) Modifiability, 2) Understandability, 3) Scalability, 4) Level of complexity. Each of these models is based on metrics extracted from the source code and the class diagrams.

SCC uses the models cited beforehand in order to calculate the software maintainability index, the metric that ARTIST will use to measure the complexity of the legacy code.

These models use several metrics to calculate maintainability as the way to calculate the complexity. In the context of ARTIST project where the feasibility for a migration to cloud is being evaluated, the maintainability metric (as defined by IEEE) for calculating the software complexity will be used:

$$\textbf{Maintenance} = 2.399 + 0.493 \times Modifiability + 0.474 \times Understability + 0.524 \times Scalability + 0.507*LOC$$

$$\textbf{Modifiability} = 0.629 + 0.471 \times NC - 0.173 \times NGen - 0.616 \times NAggH - 0.696 \times NGenH + 0.396 \times MaxDIT$$

$$\textbf{Understability} = 1.66 + 0.256 \times NC - 0.394 \times NGenH$$

$$\textbf{Scalability} = 0.182 \times 0.99 \times AC + 0.100 \times EC + 0.097 \times ND - 0.036 \times PC + 0.068 \times DMS$$

$$\textbf{LOC} = 0.269 + 0.008 \times Coupling + 0.181 \times cohesion + 0.119 \times CC + 0.084 \times ILCC$$

The required metrics to perform these models are described in [25].

The aforementioned models have been predicted using data from several sources [26] using the multivariate linear model. However the correctness and fine-tuning of the formulas have to be updated to the context of ARTIST use cases.

There are several tools available in the Open Source community that offers some of the functionalities required by SCC. A first criterion to select the list of potential candidates to be re-used has been their availability as Eclipse plugin (as the basis technology of TFT and the majority of ARTIST tools), support to Java and C# and finally the availability of the source code. Following these criteria, three existing plug-ins where analyzed in detail, Metrics [27], Sonar [28] and CodeProAnlytix [29].

After a deep analysis of these tools, all of them have been discarded as they do not accomplish the requirements for the ARTIST project, rejecting also a possible adaptation of them for platform compatibility reasons.

The current SCC prototype architecture is a java API that explores source files and UML models to generate several metrics of a specific project. It comprises three sub-components:

- Metric Explorer: This is the main component of SCC current prototype. It provides the calculation of all the required metrics that are used to generate the new ARTIST metrics. Besides, it also provides exporting features to convenient formats like XML or JSON.
- Structures: This component contains the structures of the inputs and outputs models that the Metric Explorer uses. It also provides the functionality for generating the output file formats (XML, JSON).
- Test Cases: This component is provided for testing purposes. It generates several use cases that test the functionality of the SCC generating console logs and XML files with the results.

C. TFT validation

The first validation of all the components of the TFT has been performed executing in parallel:

1. The TFT comprising the TFT-UI, component detection component, strategy suggestion component, effort estimation component and TFT repository (see Fig. 3)
2. The Software Complexity Component, which calculates the maintainability index and other required metrics per component. (see Fig 2)

The component models of the Java version of the Petstore [30] application and two ARTIST use cases, Line of Business (LoB) [31] and Distant Early Warning System

(DEWS) [31] were used as sample inputs for TFT and SCC. Petstore is a multi-tier J2EE application, a B2C Web portal that displays a Pet catalog and support basic commerce. LoB is a .NET solution over Microsoft Sharepoint [32] for collaborative business process modeling. DEWS offers a complex SOA-based system (including desktop end-user command and control UIs) enabling the early detection and warning broadcasting of tsunami threats.

The component model of Petstore and DEWS were obtained using semi-automatic MDRE techniques, but the component model of LoB was created by hand. The MDRE process followed to obtain these models was as follows. Using Modisco [33], we obtained PSMs from the legacy code. These models were abstracted to a PIM level using a search-based model exploration approach [34], using either ATL [35] query and INC-Querying [36] techniques, combined with UML profiling [37] and slicing methods [38]. A further ATL M2M transformation generated a UML component model from the UML stereotyped classes existing in the PIM, aggregating similarly stereotyped classes within the same containment (i.e. package) to constitute components.

Two sample MAT reports were used (one for each platform) as the second input of TFT. TFT was fed with the MAT report and the component model of the legacy application in order to identify suggested migration strategies for each component of the application and compute the effort estimations for these strategies. TFT triggered its expert knowledge base (encoded as a set of rules) to suggest and select migration strategies for each component located in the input model. The migration complexity reported by TFT is the average of the complexity of selected strategies (information encoded in the TFT expert knowledge base as well). The estimated migration efforts are computed by TFT following a similar FP analysis conducted in [7] as the sum of efforts computed for each strategy selected for each component.

TFT was successful to deliver meaningful results in both migration suggestions and effort computations. In order to improve the quality of the suggestions, a deeper analysis on the components and its complexity metrics is required which is achievable by creating more complex rule definitions. The migration effort computation may be enhanced by increasing the number of evaluated applications thus enlarging the historical data.

In Fig. 2 and 3 the results for DEWS use case are shown:

```
****** Maintenance
Component: org.aspencloud.widgets Maintenance: 2.6357682
Component: org.aspencloud.widgets.cdatepicker Maintenance:
2.652078
Component: org.aspencloud.widgets.cnumpad Maintenance:
2.6818948
Component: org.aspencloud.widgets.snippets Maintenance:
2.7467294
Component: org.dews_online.ccui Maintenance: 2.5707283
Component: org.dews_online.ccui.control.jobs Maintenance:
2.590284
Component: org.dews_online.ccui.splashHandlers Maintenance:
2.6118982
```

Figure 2. SCC console log for DEWS (Maintainability metric)

Component Name	Stereotypes	Strategy Suggestion	Complexity	Estimated Effort
org.dews_online.cc				
MonitoringPers	Perspective ,SWT	Migrate to GWT	AVERAGE	7
ForecastingPers	Perspective ,SWT	Migrate to GWT	AVERAGE	7
MessageCompo	Perspective ,SWT	Migrate to GWT	AVERAGE	7
MessageDissem	Perspective ,SWT	Migrate to GWT	AVERAGE	7
ADewsBusiness	Subject ,JMS	Migrate to TQA	HIGH	7
SensorNetworkI	Subject ,JMS	Migrate to TQA	HIGH	7
SensorTypeTim	Subject ,JMS	Migrate to TQA	HIGH	7
DisseminationS	Subject ,JMS	Migrate to TQA	HIGH	7
DataModelCon	DataModel ,JMS	Migrate to TQA ,M...	HIGH	13

Figure 3 TFT Inventory View showing migration suggestions and efforts for DEWS

V. CONCLUSIONS

This paper presents a systematic approach that enables an early estimation of the complexity and the efforts required for the migration of existing applications to a Cloud provider. This approach combines traditional FPA techniques for migration task decomposition and effort estimation with others such as a) model-driven reverse engineering and model comprehension techniques to capture information about application components, b) expert judgment (for task suggestion and complexity estimation) implemented as a knowledge base of domain specific heuristics and c) complexity estimation (i.e. software maintainability) using an empirical combination of computable metrics. A prototypical implementation of this approach, available as an Eclipse plugin, has been described. Preliminary evaluation of the approach and tooling support has been conducted in an early evaluation of some case studies. This have enabled us to increase the TFT knowledge base of rules suggesting migration tasks and estimating their complexity, relying on the migration experiences gained through these cases. Nonetheless, the lack of reported experiences about migrating to Cloud has constrained our knowledge base to the expert judgment acquired in these few experiments and the effort figures reported on [6]. Nonetheless, the TFT decoupling between its knowledge base and its implementations eases the extension of the knowledge base as soon as new insights are gathered in other validation experiments. Foreseen future work, in the short term, includes: a) the integration of computed SCC metrics, in the computation of migration task efforts using empirical formulas that combines component maintainability with task complexity, b) the extension of TFT knowledge base to incorporate additional expert judgment heuristics to suggest additional Cloud optimization patterns, c) adjustment of the TFT effort figures collecting experimental data from ARTIST migration case studies.

VI. ACKNOWLEDGMENT

This work has been supported by the ARTIST Project and has been partly funded by the European Commission under the Seventh (FP7 - 2007-2013) Framework Programme for Research and Technological Development, grant no. 317859.

 ISBN: 978-1-61208-367-4

VII. REFERENCES

[1] ARTIST project "Advance software based service provisioning and migration of legacy Software", http://www.artist-project.eu/ [retrieved: July, 2014].

[2] M. Shepperd,C. Schofield, "Estimating software project effort using analogies", IEEE Transactions on Software Engineering, vol. 23, no. 11, Nov. 1997, pp. 736-743.

[3] M. Jorgensen, "A review of studies on expert estimation of software development effort", Journal of Systems and Software, vol. 70, no. 1-2, 2004, pp. 37-60.

[4] M. Jorgensen and M. Shepperd, "A systematic review of software development cost estimation studies", IEEE Transactions on Software Engineering, vol. 33, no. 1, January 2007. pp. 33-53.

[5] A. Albrecht, J. Gaffney, "Software function, source lines of code, and development effort prediction: A software science validation", IEEE Transactions on Software Engineering, vol. 9, 1983, pp. 639-648.

[6] V. Tran, JW. Keung, A. Liu, A. Fekete. "Application Migration to Cloud: A Taxonomy of Critical Factors". SECLOUD 2011 Software Engineering For Cloud Computing Workshop, 2011, pp. 22-28.

[7] V. Tran, JW. Keung, A. Liu, A. Fekete ."Size Estimation of Cloud Migration Projects with Cloud Migration Point (CMP)", Proceedings of International Symposium on Empirical Software Engineering and Measurement (ESEM'11), Banff, Canada, September 2011, pp. 265-274.

[8] B. Touesnard, "Software Cost Estimation: SLOC-based Models and the Function Points Model". Version 1.1, 2004.

[9] H. Najadat, I., Alsmadi,Y., Shboul. "Predicting Software Projects Cost Estimation Based on Mining Historical Data", ISRN Software Engineering Volume 2012.

[10] H.Ramakrishnan, "Analysis of complexity and coupling metrics of subsystems in large scale software systems", M. S Thesis 2006.

[11] ARTIST Consortium "D6.2.1 ARTIST Methodology" http://www.artist-project.eu/sites/default/files/D6.2.1_ARTISTMethodology_M12_30092013.pdf [retrieved: July, 2014].

[12] ARTIST Consortium "Business and Technical Modernization assessment tool M12", http://www.artist-project.eu/sites/default/files/D5.2.1Businessand Technical Modernizationassessmenttool_M12_30092013.pdf, [retrieved: July, 2014].

[13] ARTIST Consortium "Methodology and Environment for evaluating migration success", http://www.artist-project.eu/sites/default/files/D11.3.1 Methodology and Environment for evaluating migration success M8_31052013.pdf [retrieved: July, 2014].

[14] JBoss Drools, http://www.jboss.org/drools/ [retrieved: July, 2014].

[15] ARTIST consortium, "ARTIST Integrated Architecture M15.

[16] Papyrus, http://www.eclipse.org/papyrus/, [retrieved: July, 2014].

[17] EMF /UML2 http://www.eclipse.org/modeling/mdt/?project=uml2 [retrieved: July, 2014].

[18] McCabe (1976). "A Complexity Measure". IEEE Transactions on Software Engineering http://www.literateprogramming.com/mccabe.pdf, [retrieved: July, 2014].

[19] http://metrics.sourceforge.net/ [retrieved: July, 2014].

[20] R. Martin. "OO Design Quality Metrics: An Analysis of dependencies". Workshop Pragmatic and Theoretical Directions in Object-Oriented Software Metrics. http://www.cin.ufpe.br/~alt/mestrado/oodmetrc.pdf. [retrieved: July, 2014].

[21] Instability http://en.wikipedia.org/wiki/Software_package_metrics, [retrieved: July, 2014]

[22] IEEE (1990). "IEEE Std 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology".

[23] Ch. Gautam,, S. Kang, "Comparison and implementation of software maintenance models". International Journal of Engineering Research & Technology (IJERT), Vol. 1 Issue 6, August 2012.

[24] S.W.A. Rizvi, R.A. Khan, "Maintainability Estimation Model for Object Oriented Software in Design Phase (MEMOOD)". Journal of Computing. Volume 2, Issue 4, April 2010

[25] M. Genero, M. Patiani, C. Calero, (2005)"A Survey of Metrics for UML Class Diagrams" Journal of Object Technology http://www.jot.fm/issues/issue_2005_11/article1/article1.pdf [retrieved: July, 2014]

[26] S. Muthanna, K. Kontogiannis, K. Ponnambalam, B. Stacey, "A maintainability model for industrial software systems using design level metrics". Published in IEEE Proceedings Seventh Working Conference on Reverse Engineering, pp.248-256, 2000.

[27] http://sourceforge.net/projects/metrics2/ [retrieved: July, 2014].

[28] http://docs.codehaus.org/display/SONAR/Using+Sonar+in+Eclipse [retrieved: July, 2014].

[29] https://developers.google.com/java-dev-tools/codepro/doc/ [retrieved: July, 2014]

[30] Java PetStore http://www.mia-software.com/html/miaStudio/download/modisco/examples/javapetstore-2.0-ea5.zip [retrieved: July, 2014]

[31] ARTIST Consortium "Use cases definition and migration architecture" (2012) http://www.artist-project.eu/sites/default/files/D12.1 Use Cases definition and migration architecture_M12_01102013.pdf [retrieved: July, 2014]

[32] Microsoft Sharepoint: http://office.microsoft.com/en-001/sharepoint/ [retrieved: July, 2014]

[33] H. Brunelière, J. Cabot, G. Dupé, F. Madiot. MoDisco: a Model Driven Reverse Engineering Framework.Information and Software Technology 56, 8 ,2014, pp.1012-1032

[34] P. Baker , M. Harman , K. Steinhofel , A. Skaliotis, Search Based Approaches to Component Selection and Prioritization for the Next Release Problem, Proceedings of the 22nd IEEE International Conference on Software Maintenance, p.176-185, September 24-27,

[35] F., Jouault, and I. Kurtev,:On the Architectural Alignment of ATL and QVT. In: Proceedings of ACM Symposium on Applied Computing (SAC 06), Model Transformation Track. Dijon (Bourgogne, FRA), April 2006 [retrieved: July, 2014]

[36] INC-Query, http://www.eclipse.org/incquery/

[37] J. Cabot, C. Gómez. A simple yet useful approach to implementing UML Profiles in current CASE tools. In Workshop in Software Model Engineering, 2003.

[38] A., Blouin, B., Combemale, B., Baudry, O., Beaudoux. "Modeling Model Slicers". Model Driven Engineering Languages and Systems. Lecture Notes in Computer Science Volume 6981, 2011, pp 62-76

 ISBN: 978-1-61208-367-4

Model Reverse-engineering of Mobile Applications with Exploration Strategies

Sébastien Salva
LIMOS - UMR CNRS 6158
Auvergne University, France
email: sebastien.salva@udamail.fr

Stassia R. Zafimiharisoa
LIMOS - UMR CNRS 6158
Blaise Pascal University, France
email: s.zafimiharisoa@openium.fr

***Abstract*—This paper presents a model reverse-engineering approach for mobile applications that belong to the Graphical User Interface (GUI) application category. This approach covers the interfaces of an application with automatic testing to incrementally infer a formal model expressing the navigational paths and states of the application. We propose the definition of a specialised GUI application model which stores the discovered interfaces and helps limit the application exploration. Then, we present an algorithm based upon the Ant Colony Optimisation technique which offers the possibility to parallelise the exploration and to conceive any application exploration strategy. Finally, our approach is experimented on Android applications and compared to other tools available in the literature.**

Keywords–model generation; automatic testing; android applications.

I. INTRODUCTION

Many software engineering approaches rely upon models to automate some steps of the software development life cycle. Unfortunately, these kind of approaches suffer from an indisputable problem which often makes them impractical with many real world systems: writing models, especially exhaustive ones, is often a tedious and error-prone task. As a consequence, only partial models are often available which makes model-based approaches less interesting. For instance, Model-based testing is an approach which takes formal specifications to generate test cases, but the former have to be complete.

Model inference or model reverse-engineering is a recent research field that partially address this issue. Indeed, models can be inferred from application documentation or execution traces (sequences of actions given or observed from the application) for comprehension or to automatically carry out some tasks, e.g., the test case generation. Most of the model generation approaches, available in the literature, focus on GUI applications (a.k.a. event-driven applications), which offer a Graphical User Interface (GUI) to interact with and which respond to a sequence defined by the user. In short, these applications are explored (a.k.a. crawled) with automatic testing techniques for extracting traces to derive a model. Furthermore, a large part of the application defects can eventually be detected during the process. Afterwards, these generated models may be manually extended, analysed with verification techniques or employed for generating test cases.

In this paper, we propose a model reverse-engineering approach, combined with automatic testing, which is dedicated to mobile applications. These GUI applications for smartphones, are usually poorly documented and are often manually tested. From a mobile application, our solution generates two STS (Symbolic Transition System) specifications, which can be seen as documentation either useful for maintaining the application or for comprehension, or for performing automatic model analyses and test case generation (verification with model-checkers, etc.).

Several works already deal with the crawling of GUI applications e.g., desktop applications [1], Web applications [2][3][4] or mobile ones [5][6][7]. These approaches interact with applications in an attempt to either detect bugs or record a model or both. These previous works already propose interesting features, such as the test case generation from the inferred models. Nonetheless, it also emerges that many interesting issues still remain open. Firstly, experimenting the GUIs of Web or mobile applications may lead to a large and potentially unlimited number of states that cannot be all explored. Furthermore, the application traversing is usually guided by one of these strategies: DFS (Depth First path Search) or BFS (Breadth First path Search). These are relevant on condition that all the application states would be explored. But when the application state number is large or the processing time is limited, using other strategies could help in the exploration of the most interesting features of the application as a first step.

This paper presents an innovative model generation approach which overcomes the previous problems by putting forth the following features:

- model definition and compactness: we propose an original model definition specialised to mobile applications. Combined with our application exploration algorithm, this model especially offers the advantage to help limit the exploration and to prevent from a state space explosion. But, this model can still store the discovered interfaces and their properties instead of resorting abstract event-based descriptions only. These detailed information are particularly relevant to later perform precise analyses. A bisimulation minimisation technique is also applied to yield a second reduced STS which can be more easily interpreted,
- test data generation: instead of using random test values, the values used to fulfil the application interfaces are constructed from several data sets, and in particular from a set of fake identities. Furthermore, for one interface, the set of test value tuples are constructed

 ISBN: 978-1-61208-367-4

by means of a Pairwise technique to reduce the testing cost,

- strategy choice: the application exploration is here guided by strategies that are applied on the model under generation by means of the Ant Colony Optimisation (ACO) technique. We also show that our exploration algorithm, based upon the ACO heuristic, is highly parallelisable.

The paper is structured as follows: Section II sets down the terminology of mobile applications used throughout the paper and particularly presents our model definition. We present, in Section III, our mobile application exploration algorithm based upon the ACO heuristic. We give some experimental results and compare our approach with available tools in Section IV. We briefly present some related work and discuss about our proposal in Section V and we conclude in Section VI.

II. MOBILE APPLICATION MODELLING

A. Terminology

We say that a mobile application displays (graphical user) interfaces, each representing one application state (the number of states being potentially infinite). An interface is generated by a component of the application. Here, we take back the notation used in the Android Operating System (OS) where such a component is called an *Activity*. These instantiate *Widgets* (buttons, text fields, etc.) and declare the available events that may be triggered by the user (click, swipe, etc.). A Widget is characterised by a set of properties (colour, text values, etc.); some of them are said *editable*, which means that their values can be provided by users at runtime.

We take as example the *Ebay Mobile* application, which is available on the *Google Play* store[8]. Since this complex application owns 135 Activities, we only depict a part of its storyboard in Figure 1. The launcher interface is loaded by the first Activity *eBay* (i0). A user may choose to search for an item by clicking on the editable text field Widget. In this case, the Activity *MainSearchActivity* is reached (i1). For instance, if the user enters the keyword "shoes", the search result list is displayed (i2); the Activity is unchanged. Then, three new Activities may be reached: 1) an Activity called *SegmentSearchResultActivity* (i3) displays a result when one element of the proposed list is chosen, 2) a *Scanner* Activity is started when the text field "Scan" is clicked (i4) and 3) a log-in process is performed when the "saved searches" item is selected (Activity *SignInActivity*, i5).

B. The STS model

To represent the behaviours of mobile applications, we shall consider the Symbolic Transition System (STS) model, which is a kind of automata model extended with variables that encode the state of the system. Transitions also carry actions combined with parameters, guards and assignments. We chose the STS definition proposed in [9] which does not explicitly represent states in transitions. Instead, *(control) locations* are encoded with variables taking values in finite domains. This definition offers more flexibility to represent locations that have a precise meaning by means of variables.

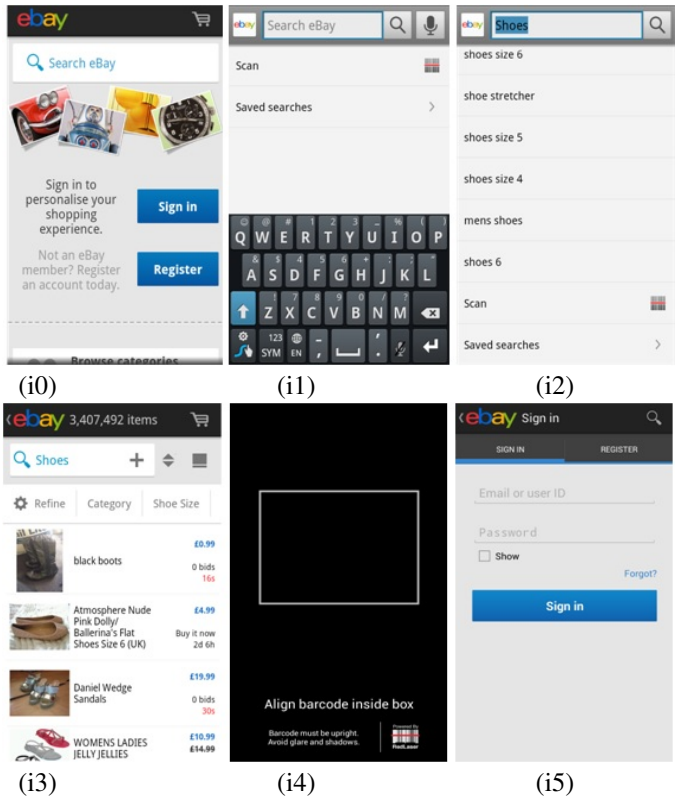


Figure 1. Ebay Mobile Storyboard

Definition 1 (STS) *A STS $\mathcal{S}$ is a tuple $< V, V0, I, \Lambda, \rightarrow >$, where:*

- *V is the finite set of internal variables and I is the finite set of parameter variables. A variable can have a simple type (Integer, String, etc.) or a complex type (List, etc.). We denote D_v the domain in which a variable v takes values. The internal variables are initialised with the initial condition $V0 \subseteq D_v$, which is assumed to be unique,*
- *Λ is the finite set of symbolic actions $a(p)$, with $p = (p_1, ..., p_k)$ a finite list of parameters in $I^k (k \in \mathbb{N})$,*
- *$\rightarrow$ is the finite transition set. A transition $(a(p), G(p, v, T(v, p)), A(v, p, T(v, p)))$ is labelled by an action $a(p) \in \Lambda$. $G \subseteq D_p \times D_V \times D_{T(p \cup V)}$ is a guard on internal variables, parameters and $T(p \cup V)$ a set of functions that return boolean values only (a.k.a. predicates) over $p \cup V$. Internal variables are updated with the assignment function $A : D_V \times D_p \times D_{T(p \cup V)} \rightarrow D_V$ once the transition is fired.*

Below, we adapt this generalised STS definition to express mobile application properties, i.e., interfaces and events.

C. Mobile application modelling with STS

We propose a STS-based model definition allowing to stock complete mobile application interfaces to yield rich models, which may be analysed afterwards. Nonetheless, a GUI application may produce a potentially infinite set of interfaces

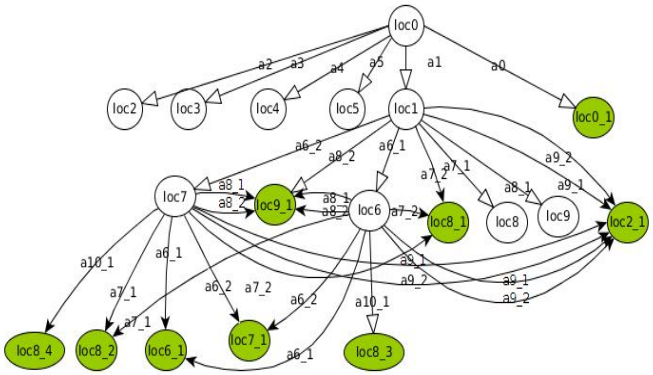


Figure 2. Ebay application STS tree

[6][10] and may lead to a state space explosion problem. We propose to get around this strong issue by focusing on the following idea: many of these interfaces are almost identical in term of content and often display different text field values. For a set of almost identical interfaces, we propose to only explore one interface in this set. To this end, we express an interface by the tuple (wp, wt) where wt is the list of Widget properties related to the text field values found in the interface and wp the remaining list of Widget properties. We define that a STS location is encoded by the variable loc, and captures a value list of the form (act, wp, wt, end, ph) with act an Activity name (or URI), accompanied by the Widget property lists wp and wt. Furthermore, these locations are completed with a boolean value denoted end indicating whether the application has to be explored from this location. Finally, the positive value ph denotes a pheromone amount that shall be used by apply the ACO technique. The purpose of this value is explained in the next Section.

We also interact with mobile applications by means of events, e.g., a click, applied on Widgets. Some editable Widgets are eventually completed before triggering the event. We capture these events with STS transitions of the form $(event(widget), G, A)$. The guard G is composed of conjunctions which show the initial location of the transition, a constraint over editable Widgets expressing their completion with user values, and the value of $widget$, giving the Widget name on which is applied the event. The assignment A gives the final location of the transition. It results that we express the functioning of a mobile application with the following STS model, called the STS Tree of an application:

Definition 2 *A mobile application is modelled by the STS Tree* $< V, V0, I, \Lambda, \rightarrow >$ *where:*

- Λ *gathers the available actions of the form* $event(widget)$*,*
- $\rightarrow$ *is composed of transitions* $(event(widget), G, A)$ *with a guard* G *of the form* $[loc == (act, wp, wt, end, ph) \wedge editable\_constraint \wedge Widget == wn]$ *and an assignment* A *of the form* $loc := (act2, wp2, wt2, end2, ph2)$*:*
  - *the expression* $loc == (act, wp, wt, end, ph)$ *gives the initial location of the transition, while the assignment* $loc := (act2, wp2, wt2, end2, ph2)$ *gives the final location.* act *is an Activity name,* wt *is a list of Widget properties relative to text field values,* wp *is a list of Widget properties excluding* wt*,* end *and* ph *are boolean values,*
  - *editable_constraint is a conjunction of atomic expressions of the form* $widgetprop == v$ *with* v *a value and the variable* $widgetprop$ *corresponding to an editable Widget property.*
  - $widget == wn$ *denotes the Widget name on which is applied the event.*
- $V0$ *denotes the initialisation of the* loc *variable.*

TABLE I. Actions and Guards of the STS Tree

Label	Action[Guard]
$a1$	click(widget)[widget=id/$home\_search\_text$]
$a6\_1$	click(widget)[widget=$id/up \wedge search\_src\_text$=All shoes]
$a6\_2$	click(widget)[widget=$id/up \wedge search\_src\_text$=shoes]
$a7\_1$	click(widget)[widget=$id/search\_button \wedge search\_src\_text$=All shoes]
$a7\_2$	click(widget)[widget=$id/search\_button \wedge search\_src\_text$=shoes]
$a8\_1$	click(widget)[widget=$id/text1 \wedge search\_src\_text$=All shoes]
$a8\_2$	click(widget)[widget=$id/text1 \wedge search\_src\_text$=shoes]
$a9\_1$	click(widget)[widget=$id/text2 \wedge search\_src\_text$=All shoes]
$a9\_2$	click(widget)[widget=$id/text2 \wedge search\_src\_text$=shoes]
$a10\_1$	click(widget)[widget=$listview$ at position 1 $\wedge search\_src\_text$=shoes]

Figure 2 illustrates an example of STS Tree derived from the Ebay Mobile application. For readability, the locations are not given in the transitions but some of them are presented in a reduced form in Table II: we give the Activity name, the numbers of Widget properties (wt and wp), and the values end and ph. The actions and guards are showed in Table I. The STS Tree is composed of several "click" actions applied on different Widgets (buttons, elements of listView Widgets, etc.). The location $loc0$ represents the initial interface *eBay* of the application, which includes 2 buttons, 6 images and 16 text fields. $loc1$ is reached from $loc0$ by executing the action $a1$, i.e., by clicking on the $home\_search\_text$ Widget. The locations $loc6$ and $loc7$ are respectively reached after the completion of the $search\_src\_text$ Widget with the "All shoes" resp. "shoes" text values and the click on the Widget up (actions $a6\_1$ or $a6\_2$). These two locations correspond to two different interfaces which differ from each other on the value of the $search\_src\_text$ field and on the Widget *listview* which is a Widget container: the latter has 1 element for $loc6$ and 10 elements for $loc7$. The locations $loc8\_i(1 \leq i \leq 4)$, reached from $loc6$ or $loc7$, express 4 interfaces which only differ from the interface stored in $loc8$ by some text field values. As a consequence, they are marked by end to stop the exploration.

After covering only 5% of the *Ebay Mobile* Activities, we already obtain 19 locations in the STS Tree. This is why our approach, explained below, relies upon a minimisation technique to reduce this location number.

III. AUTOMATIC TESTING AND MODEL GENERATION WITH ACO

Intuitively, many inference model methods consist in analysing and completing interfaces with random test data and triggering events to discover new interfaces that are recursively explored in an in-depth manner. As a consequence, the application exploration is usually guided with either a

TABLE II. Summary of some locations of the STS Tree

Loc	act	#wp	#wt	end	ph
$loc0$	eBay	2b,6im	16t	false	0
$loc1$	MainSearchAct	1b,4im	3t,1e	false	1
$loc6$	MainSearchAct	1b,3im,1l_e	3t,1e	false	2
$loc6\_1$	MainSearchAct	1b,3im,1l_e	3t,1e	true	3
$loc7$	MainSearchAct	1b,3im,10l_e	3t,1e	false	2
$loc7\_1$	MainSearchAct	1b,3im,10l_e	3t,1e	true	3

b: button e: editable text field t: text field im: image
l_e: # elements in the listview Widget

DFS (Depth First path Search) or a BFS (Breadth First path Search) strategy. Nonetheless, when an application returns a high number of new interfaces, the graph to be explored may become too large to visit in a reasonable time delay. The search is only performed to a limited depth, and the explored part of the application is not necessarily the most interesting one. In this section, we address this issue and we propose an algorithm which includes the possibility to define an exploration strategy.

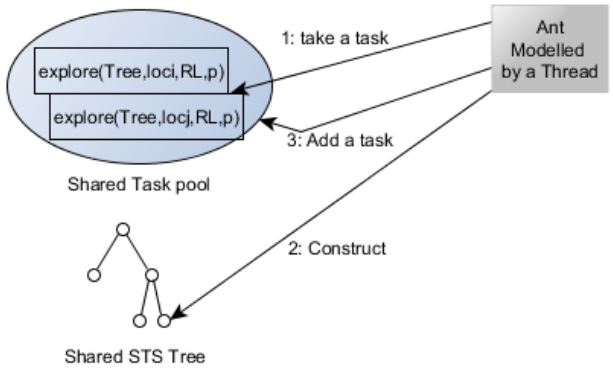


Figure 3. Parallel exploration functioning

Our proposal applies strategies by means of the Ant Colony Optimisation (ACO) technique. With ACO, the optimal path search in a graph is performed by simulating the behaviour of ants seeking a path between their colony and a source of food: firstly, the ants explore randomly and lay down little by little pheromone trails that are finally followed by all the ants. In our case, this solution leads to the architecture illustrated in Figure 3. The STS construction is guided by laying down in locations an amount of pheromone with regards to the chosen strategy. Each location exploration is considered as a task that is placed into a task pool, implemented as an ordered list, and executed by threads simulating ants. Then, our algorithm proceeds by exploring first the locations having the highest pheromone amount. The process ends when the task pool is empty. These steps are explained below:

A. Application exploration

Algorithm 1 takes as input a mobile application app and starts it to analyse its first interface and to initialise the first location loc_0 of the STS Tree $Tree$. This step is carried out by one thread only. The analysis of an interface does not rise any technical difficulty with mobile applications (Android and iOS). Indeed, it is always possible to retrieve the Activity and the Widget properties of the current interface with testing tools such as Robotium [11]. Afterwards, the interface exploration can begin: each thread (ant) executes the loop of

Algorithm 1: Mobile application exploration simplified Algorithm

input : Application app
output: STS $Tree$, $MTree$

```
// initialisation performed by one ant only
Start the application app;
Analyse the current interface → Activity act, the Widget property lists wp, wt;
Initialise ph_0 (depends of the chosen strategy);
loc_0:= (act, wp, wt, false, ph_0);
Initialise STS Tree (V0_Tree = loc_0);
Add (Explore(Tree, loc_0, RL = {(act, wp)}, p = ∅)) to the task pool;
// code performed by all the ants
while the task pool is not empty do
    Take a task (Explore(Tree, loc_i, RL, p)) having a location (act, wp, wt, end, ph) with the highest pheromone amount ph;
    Reset and Execute app by covering p;
    Explore(Tree,loc_i,RL,p);
// code performed by one ant only
MTree:= Minimise(Tree);
```

TABLE III. Location blocks of the minimised STS Tree

block	locations
$B1$	$loc6, loc7$
$B2$	$loc6\_1, loc7\_1$
$B3$	$loc8\_1, loc8\_2, loc8\_3, loc8\_4$

Algorithm 1: while there is a task to do, an instance of the application is launched in a re-initialised test environment and a task $(Explore(Tree, loc_i, RL, p))$ having a location loc_i, composed of the highest pheromone amount, is picked out. This task aims at exploring one interface only and may produce other tasks. The set RL, used by Explore, stores the discovered locations in a reduced form (act, wp).

After the end of the exploration, a second STS $MTree$ is computed with a minimisation technique. The STS minimisation aims to yield a more compact STS in term of STS location number and more readable for application comprehension. Here, we have chosen a bisimulation minimisation technique since this one preserves the functional behaviours represented in the original model. The time complexity of this minimisation technique is also reasonable (proportional to $\mathcal{O}(mlog(n))$ with m the transition number and n the state number). A detailed algorithm can be found in [12]. This algorithm constructs the location sets (blocks) that are bisimilar equivalent. Due to lack of room, we only present in Figure 4 and in Table III, the minimised STS obtained from the STS Tree of Figure 2. Some locations are now grouped into blocks: for instance, the locations $loc6$ and $loc7$ are grouped into the Block $B1$ because the same action sequences leading to bisimilar locations can be executed from both $loc6$ and $loc7$.

One task, pulled from the task pool, is now performed by calling the Explore procedure given in Algorithm 2. It takes the STS under construction $Tree$, a location loc_i, a path p and the set RL of discovered locations stored in a reduced form. Initially, the procedure ends if a stopping condition, based upon the code coverage and on the processing time, holds. This condition allows to stop the exploration after a reasonable time delay. Otherwise, the Explore procedure calls *GenConstraints* to analyse the current interface, extract the editable Widgets

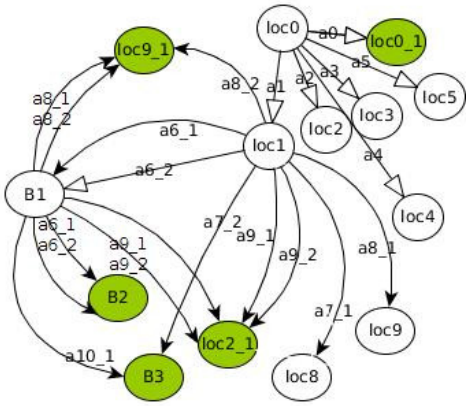


Figure 4. Minimised STS Tree

and to produce a set of constraints expressing how fulfilling these editable Widgets with test values. Similarly, the events that can be triggered on the Widgets are dynamically detected (with testing tools). It results a set of couples $(event, w)$ with $event$ the event to apply on the Widget w. Then, the exploration of the current interface begins. Its editable Widgets are completed in accordance with a constraint c (line 7). A Widget w is stimulated with an event in reference to a couple $(event, w)$ found in the Events set. This results in a new interface $Inew$ (line 9). A $Ph\_Deposit$ procedure is called to compute the pheromone amount that shall be deposed in the arrival location of the transition constructed in the next step. This amount is computed with regards to the chosen strategy. The algorithm now checks whether this interface and its corresponding location have to be explored. Naturally, if $Inew$ reflects the end of the application (exception, crash), $Inew$ must not be explored. Furthermore, if $Inew$ only differs from a previously encountered interface by its text field values, we also stop the exploration. This is done in the algorithm by checking if the list (act_j, wp_j), extracted from $Inew$, which excludes the Widget properties related to text field values, belongs to the set RL. If one of these conditions hold then a new transition carrying the arrival location $(act_j, wp_j, wt_j, true, ph_j)$ is added to the STS $Tree$. The boolean value $true$ denotes that this location must not be explored. On the contrary, a new transition is added (with a location loc_j whose last boolean value is set to false). This location loc_j has to be explored. Therefore, a new task is added to the task pool (line 18). To apply the next constraint and event, the application has to go back to its previous interface by undoing the previous interaction. This is done with the Backtrack procedure (line 20) whose role is to undo the most recent action. When the direct interface restoration is not possible (when the backtrack mechanism is not implemented or when the application crashed), the Backtrack procedure resets the application and incrementally replays the actions of the path p.

This algorithm also relies upon the procedure *GenConstraints* to construct constraints expressing how to fulfil an interface under test with values. Due to lack of room, we present it succinctly. The *GenConstraints* procedure aims to generate constraints of the form $w_1.value = v1 \wedge$ ... $\wedge w_n.value = vn$, with $(w_1, ..., w_n)$ the list of editable Widgets of an interface and $(v_1, ..., v_n)$, a list of test values. Instead of using random values like in many model inference approaches, we propose

Algorithm 2: Explore Procedure

```
1  Procedure Explore(Tree, loc_i, RL, p);
2  if [processing time > T or code coverage > CC] then
3      stop;
4  Generate constraints with GenConstraints → C;
5  Analyse the current interface → Events;
6  foreach c ∈ C ∧ (event, w) ∈ Events do
7      fulfil the editable Widgets with c;
8      Apply event on the Widget w → new interface Inew;
9      Analyse the interface Inew → act_j, wp_j, wt_j;
10     ph_j := Ph_Deposit(loc_i, act_j, wp_j, wt_j);
11     if Inew is empty or Inew reflects a crash or there exists
       (act_j, wp_j) ∈ RL then
12         {Add a transition (event(widget),
           G = [loc == loc_i ∧ c ∧ widget == w], A = (loc :=
           (act_j, wp_j, wt_j, true, ph_j))) to →_Tree;
13         } (in critical section)
14     else
15         loc_j := (act_j, wp_j, wt_j, false, ph_j);
16         {Add a transition t = (event(widget),
           G = [loc == loc_i∧c∧widget == w], A = (loc := loc_j))
           to →_Tree;
17         RL := RL ∪ {(act_j, wp_j)}
18         Add the task (Explore(Tree, loc_j, RL, p.t)) to the task
           pool;
19         } (in critical section)
20     Backtrack(loc_i, p);
```

to use several data sets: a set $User$ of values, eventually composed of logins and passwords, provided by a user, a set RV composed of values well known for detecting bugs, e.g., String values like "&", "", or null. A last set, denoted $Fakedata$, is composed of fake user identities. An identity is itself a list of parameters $(p_1, ..., p_m)$, such as (name, age, email, address, gender), that are correlated together to form realistic identities. Both $User$ and RV sets are segmented per type (String, Integer, etc.). We denote $type(User \cup RV) \subset User \cup RV$ the subset of values having the type $type$. The *GenConstraints* procedure starts collecting the editable Widget list $(w_1, ..., w_n)$. Every w_i is then associated to a specific data set as follows:

1) *GenConstraints* extracts the larger subset $(w_1, ..., w_k)$ which is also a subset of the parameter list $(p_1, ..., p_m)$ of $Fakedata$ (we try to find a correlation between the Widget names and the identity parameters with regular expressions). This subset of Widgets is then associated to a list of "reduced" identities where the parameters which do not belong to $(w_1, ..., w_k)$ are removed. For instance, if two Widgets called *name* and *email* are found, the fake identities of $Fakedata$ are parsed to remove the undesired parameters and to return the set of identities composed only of a name and an email,
2) each remaining Widget, is associated to the value set $t(User \cup RV)$ with t the type of data expected by the Widget (usually String). We obtain a list of value sets $\{V_1, ..., V_n\}$ linked to the Widgets $(w_1, ..., w_n)$

Now, instead of using a cartesian product to derive a set of tuple of values denoted V, we adopted a Pairwise technique [13]. Assuming that errors can be revealed by modifying pairs of variables, this technique strongly reduces the coverage of variable domains by constructing discrete combinations for pair of parameters only. Finally, the set of constraints C is derived from V.

 ISBN: 978-1-61208-367-4

Last but not least, our proposal also offers the advantage of being highly parallelisable. Indeed, the task pool is a known paradigm of parallel computing where the tasks of the pool are executed in parallel on condition that the tasks are independent. This is the case in our Algorithms since several application instances are experimented into independent test environments. All the threads share the same STS $Tree$, the same discovered location set RL and the same task pool implemented as an ordered list. This is why we added three critical sections in the Explore procedure to prevent concurrent accesses when transitions are added to the STS (lines 12,13,16), or when a task is added into the pool (line 18).

Complexity and termination of Algorithm 1: theoretically, this algorithm does not end if the number of new interfaces to visit is infinite. This is why we added a stopping condition in the Explore procedure. But, our algorithm only explores the interfaces, which have new Widget properties (in excluding those related to text field values), and we have observed in practice that the number of these interfaces is often bounded. Consequently, our algorithm ends with most of the applications. If we assume that the number of locations to visit is then bounded to n, Algorithm 1 has a complexity proportional to $\mathcal{O}(m+n+mn+2mlog(n))$ with m the number of transitions.

B. Exploration strategies

Different strategies can be now used to cover an application. We succinctly present some of them below. These have to be implemented in the $Ph\_Deposit$ procedure.

- **DFS-BSF strategy:** a combination of both DFS and BFS strategies can be easily put into practice as follows: the location loc_0 is initialised with a pheromone amount equal to 0. Afterwards, whenever a new location loc_j is detected from an initial one loc_i, it is completed with the pheromone amount found in loc_i increased by 1. In this case, the next task chosen in the task pool shall be the one including the first discovered location from loc_i. Tacitly, a DFS strategy is followed. But, the current location being explored, is also completely covered in a breadth-wise order first,
- **crash-driven strategy:** the number of observed crashes could also be considered in a strategy: when the number of crashes detected from the locations of a path p is higher than the crash number detected from the locations of another path p', it may be more interesting to continue to cover the former for trying to detect the highest number of crashes. We call this strategy crash-driven exploration. This can be conducted as follows: the pheromone amount is initialised to 0 in loc_0. Whenever a new location loc_j is built, it is completed with a pheromone amount equal to the addition of the pheromone amount found in the preceding location loc_i with the number of crashes (or exceptions) detected from loc_i,
- **semantics-driven strategy:** these strategies denote an exploration guided by the recognition of the meaning of some Widget properties (text field values, etc.). Here, the pheromone deposit mainly depends on the number of recognised Widget properties and on their relevance. It is manifest that the semantic-driven strategy domain can be tremendously vast. For e-commerce applications, the login step and the term "buy" are usually important. A strategy example could be then conducted as follows: an authentication process is detected when a text field Widget has the type "passwdtype". In this case, the pheromone amount considered is set to X, otherwise it is equal to 1. When a Widget name is composed of the term "buy", the pheromone amount added in the location could be $Y < X$, etc.

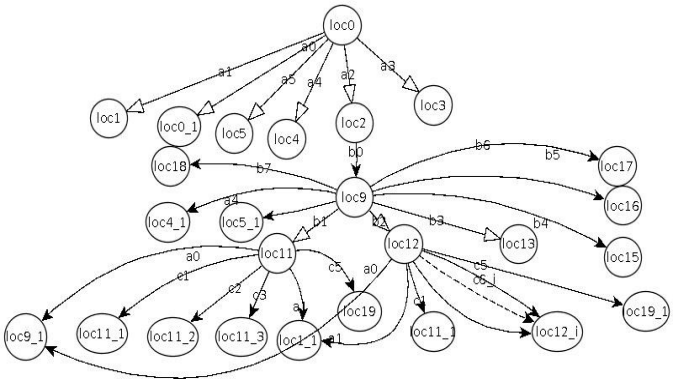


Figure 5. Ebay Mobile STS Tree obtained with a semantics-driven strategy

Many other strategies could be defined in relation to the desired result in terms of model generation and test coverage. Other criteria, e.g., the number of Widgets, could also be taken into consideration. The strategies, succinctly described above, could also be mixed together.

The STS Tree of Figure 2 is constructed with Algorithm 1 related to the DFS-BFS strategy. The Explore procedure starts the exploration from $loc0$, which holds a pheromone amount equal to 0. The actions $a0$ to $a5$ lead to new interfaces and locations $loc1, ..., loc5$ that have to be explored. Here, the location $loc1$ is chosen since it is the first new encountered location and has the highest pheromone amount. From $loc1$, the execution of actions leads to new locations: for instance the locations $loc8$ and $loc8\_1$ are reached with the actions $a7\_1$ and $a7\_2$. These locations only differ by their text field values. Hence, the arrival location $loc8\_1$ is not explored and marked by end. The next location having the highest pheromone amount is $loc6$. Therefore, this one is explored. And so on.

We also applied a semantics-driven strategy on this application to illustrate the different STS Trees which may be generated. This strategy aims to target the account management part of the application and was applied by deposing a higher pheromone amount in locations including Widgets of type "passwdtype" or Widget properties composed of the terms "account" or "sign in". Figure 5 illustrates the resulting STS Tree after applying this strategy: here the Activity $SignInActivity$ ($loc2$), allowing to manage user accounts, was targeted instead of the Activity $MainSearchAct$ ($loc1$). This strategy makes the generated STS more interesting to later analyse the security of the application or to generate security test cases.

 ISBN: 978-1-61208-367-4

TABLE IV. Processing time to explore all the locations with different strategies

Application	DFS(1)	DFS-BFS(1)	DFS-BFS(3)
Converter	478	435	295
NotePad	268	310	175
Tippy Tipper	251	210	110
ToDoManager	551	410	210
LotsA	70	83	48
OpenManager	696	560	489
HelloAUT	106	216	201
TomDroid	235	256	196
ContactManager	233	216	135
OpenSudoku	434	456	411

TABLE V. Code and Activity coverage

Applications	Monkey	Orbit	GUI TAR	GUI Ripper	MCrawlT	
					Code cov.	Act. cov.
NotePad	60	82	-	-	88	100
ToDoManager	71	75	71	-	81	100
HelloAUT	71	86	51	-	96	100
TomDroid	46	70	-	40	76	100
Youtube	-	-	-	-	-	54.5
CNN	-	-	-	-	-	73
TaskKiller	-	-	-	-	-	57.1
Ebay	-	-	-	-	-	19
WordPress	-	-	-	39	-	47
CatLog	-	-	-	-	77	80
DiskToFon	-	-	-	-	42	67
SipDroid	-	-	-	-	-	11

IV. EXPERIMENTATIONS

We conducted several empirical studies to assess the overall results of our approach applied on Android mobile applications. Our prototype tool, called *MCrawlT (Mobile Crawler Tool)*, is publicly available in a Github repository [14]. It takes packaged Android applications (apk files) or source projects and stimulates them by calling the testing framework *Robotium*. This one is also employed to analyse interfaces. An application can be experimented in parallel by launching several Android emulators. We randomly chose some Android applications of the *Google Play* store and some applications taken as examples in other papers dealing with Android application automated testing for comparison purposes.

Table IV reports the processing time required for completely exploring these applications with a Mid 2011 computer including a CPU 2.1Ghz Core i5 and 4GB of RAM. The tool were applied with a DFS strategy (1 emulator), a mixed DFS-BFS (with 1 and 3 emulators in parallel). Our results firstly show that the chosen strategy has a direct impact on the processing time required to cover an application. In this experimentation, half of the applications are more rapidly covered with DFS-BFS traversing. For instance, with *toDoManager*, using a DFS-BFS strategy instead of a DFS one, reduces the exploration delay by 140 seconds because all of its Activities are directly accessible from the initial one. These results depend mainly on the application structure though. When the insight of the application structure is known, our tool offers the advantage of choosing the most appropriate strategy. Table IV also shows that the parallelisation of our algorithm is effective. With three emulators, the processing time is always reduced. For instance, the parallel exploration of *Tippy Tipper* is achieved with a processing time almost divided by two.

TABLE VI. Crash detection

Applications	MCrawlerT	Monkey	GUI Ripper
Converter	9	4	
Notepad	2		
TomDroid	3	1	14
WordPress	51	3	37
CatLog	17	0	
DiskToFon	2	0	
Sipdroid	1	1	

Table V shows the resulting code coverage obtained with our tool and other crawlers available in the literature: Monkey [15], Orbit [7], GUITAR [1], GUI Ripper [6]. With our tool, we provide the code coverage that is obtained for the applications whose source code is available (small open source applications). For the others, we can only give the Activity coverage (explored Activities). Most of the other tools explore Android applications in an in-depth manner. Therefore, MCrawlT was executed only with this strategy to carry out a fair comparison. These results show that the code coverage is between 42% and 96%. An application is incompletely covered either on account of unused code parts (libraries, packages, etc.) that are not called, or on account of functionalities difficult to start automatically. The code coverage achieved with MCrawlT is either equivalent or higher than the one given by the other tools. For instance with *TomDroid*, we obtained 76 %, whereas ORBIT covers 70%, Monkey 46% and GUI Ripper 40% of the code. ORBIT offers a better code coverage with *Contactmanager* though. Indeed, users interact on this application with long click events that are supported by Orbit but not yet by our tool. The last lines of Table V show the results obtained with larger applications (*Youtube* to *Sipdroid*). Since the time required to discover these applications may be long, we have limited the exploration time to 30 minutes. Without limitation, the coverage should strongly augment. Surprisingly, this kind of application is not considered by the other tools.

Finally, Table VI illustrates the number of observed crashes on some Android applications with our tool *MCrawlT*, *Monkey* and *GUI Ripper*. We only show the applications for which at least one error has been detected with one of the tools. The processing time was limited to 30 minutes for the two first tools. For GUI Ripper, we have taken back the experimental results given in [6] that were obtained with a processing time varying between 3 and 5 hours. *MCrawlT* outperforms *Monkey* in automatic crash detection, which is not surprising since the former covers deeper Android applications. The comparison with GUI Ripper is less obvious since the authors only provide two detailed results with this tool. For *WordPress*, *MCrawlT* detects more crashes than *GUI Ripper*, and on the contrary, more crashes are detected with *TomDroid*. But, the processing time of GUI Ripper is twelve times more long.

All these experimental results on real applications tend to show that our tool is effective and leads to substantial improvement in the automatic testing and model inference of GUI applications.

V. RELATED WORK AND DISCUSSION

Several papers dealing with automatic testing and model generation approaches of black-box systems were issued in the last decade. Here, we present some of them relative to our work:

 ISBN: 978-1-61208-367-4

Memon et al. [1] initially presented GUITAR, a tool for scanning desktop applications. This tool produces event flow graphs and trees showing the GUI execution behaviours. Only the click event can be applied and GUITAR produces many false event sequences which may need to be weeded out later. Furthermore, the actions provided in the generated models are quite simple (no parameters). Mesbah et al. [2] proposed the tool Crawljax specialised in Ajax applications. It produces a state machine model to capture the changes of DOM structures of the HTML documents by means of events (click, mouseover, etc.). To limit the state space and to avoid a state explosion problem, state abstractions should be given manually to extract a model with a manageable size. The concatenation of identical states proposed in [2] is done in our work by minimisation.

Google's Monkey [15] is a random testing tool (events and data) offering light coverage especially when an authentication is required in the application. No model is provided. Amalfitano et al. [6] proposed GUI Ripper, a crawler for crash testing and for regression test case generation. A simple model, called GUI tree, depicts the observed GUI. Then, paths of the tree not terminated by a crash detection, are used to re-generate regression test cases. Joorabch et al. [10] proposed another crawler, similar to GUI Ripper, dedicated to iOS applications. In comparison to these works, our generated models are much more detailed and can be used to derive new test cases since all the actions and parameters can directly be found in STS Trees. We also consider several exploration strategies. The novelty of the work proposed by Yang et al. [7] lies in the static analysis of Android application codes to infer the events that can be applied to the GUI. Then, a classical crawling technique is employed to derive a tree composed of events. This grey-box testing approach was implemented in the Orbit tool. When only one emulator is used, this approach should cover an application quicker than our proposal since the events to trigger are listed by the static analysis whereas we try to detect them dynamically. But, Orbit can be applied only when the application source code is available. This is not the case for many Android applications. Furthermore, our tool should cover an application quicker than Orbit since the former can be experimented in parallel with several emulators. Another strong advantage proposed in our approach, is the support of different exploration strategies. These can reduce the exploration time when the application structure is known or can guide the exploration when the application interface number is large.

VI. Conclusion

This paper presents a formal model inference approach for mobile applications, which performs automatic testing through application interfaces and which explores applications by means of strategies. For one application, two STS models are generated by this approach. Both express the functional behaviours of the application, but the second one is reduced with a bisimulation technique for readability.

In comparison to the application crawlers available in the literature, this approach takes another direction by proposing the two following main contributions. We propose a formal model definition whose aims are to store rich details about the encountered interfaces and to help reduce the application exploration. Our algorithms are based upon the application of the ACO technique to guide the application exploration with strategies that can be modified by managing differently the pheromone deposit in locations. Our experimental results show that this approach can be used in practice: the prototype tool provides a good application code coverage in a reasonable time delay.

Acknowledgment

This reasearch was conducted with the support of the " Digital Trust" Chair from the University of Auvergne Foundation.

References

[1] A. Memon, I. Banerjee, and A. Nagarajan, "Gui ripping: Reverse engineering of graphical user interfaces for testing," in Proceedings of the 10th Working Conference on Reverse Engineering, ser. WCRE '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 260–269.

[2] A. Mesbah, A. van Deursen, and S. Lenselink, "Crawling Ajax-based web applications through dynamic analysis of user interface state changes," ACM Transactions on the Web (TWEB), vol. 6, no. 1, 2012, pp. 1–30.

[3] S. Artzi, A. Kiezun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. Ernst, "Finding bugs in web applications using dynamic test generation and explicit-state model checking," Software Engineering, IEEE Transactions on, vol. 36, no. 4, 2010, pp. 474–494.

[4] V. Dallmeier, M. Burger, T. Orth, and A. Zeller, "Webmate: a tool for testing web 2.0 applications," in Proceedings of the Workshop on JavaScript Tools, ser. JSTools '12. New York, NY, USA: ACM, 2012, pp. 11–15.

[5] S. Anand, M. Naik, M. J. Harrold, and H. Yang, "Automated concolic testing of smartphone apps," in Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, ser. FSE '12. New York, NY, USA: ACM, 2012, pp. 1–11.

[6] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, and A. M. Memon, "Using gui ripping for automated testing of android applications," in Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, ser. ASE 2012. New York, NY, USA: ACM, 2012, pp. 258–261.

[7] W. Yang, M. R. Prasad, and T. Xie, "A grey-box approach for automated gui-model generation of mobile applications," in Proceedings of the 16th international conference on Fundamental Approaches to Software Engineering, ser. FASE'13. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 250–265.

[8] Google, "Android google play store," last accessed August 2014. [Online]. Available: https://play.google.com/store

[9] T. Jéron, "Symbolic model-based test selection," Electronic Notes in Theoretical Computer Science, vol. 240, no. 0, 2009, pp. 167 – 184. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S157106610900173X

[10] M. E. Joorabchi and A. Mesbah, "Reverse engineering ios mobile applications," in Proceedings of the 2012 19th Working Conference on Reverse Engineering, ser. WCRE '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 177–186.

[11] Robotium, "Robotium tool," last accessed August 2014. [Online]. Available: https://code.google.com/p/robotium/

[12] J.-C. Fernandez, "An implementation of an efficient algorithm for bisimulation equivalence," Science of Computer Programming, vol. 13, 1989, pp. 13–219.

[13] M. B. Cohen, P. B. Gibbons, W. B. Mugridge, and C. J. Colbourn, "Constructing test suites for interaction testing," in Proc. of the 25th International Conference on Software Engineering, 2003, pp. 38–48.

[14] MCrawlerT, "Mobile crawler tool," last accessed August 2014. [Online]. Available: https://github.com/statops/MCrawlerT

[15] Google, "Ui/application exerciser monkey," last accessed August 2014. [Online]. Available: http://developer.android.com/tools/help/monkey.html

 ISBN: 978-1-61208-367-4

On the Ability of Functional Size Measurement Methods to Size Complex Software Applications

Luigi Lavazza Sandro Morasca Davide Tosi
Dipartimento di Scienze Teoriche e Applicate
Università degli Studi dell'Insubria
Varese, Italy
{luigi.lavazza, sandro.morasca, davide.tosi}@uninsubria.it

***Abstract*—The most popular Functional Size Measurement methods, namely IFPUG Function Point Analysis and the COSMIC method, adopt a concept of "functionality" that is based mainly on the data involved in functions and data movements. Neither of the mentioned methods takes directly into consideration the amount of data processing involved in a process. Functional size measures are often used as a basis for estimating the effort required for software development, and it is known that development effort does depend on the amount of data processing code to be written. Thus, it is interesting to investigate to what extent the most popular functional size measures represent the functional processing features of requirements and, consequently, the amount of data processing code to be written. To this end, we consider a few applications that provide similar functionality, but require different amounts of data processing. These applications are then measured via both functional size measurement methods and traditional size measures (such as Lines of Code). A comparison of the obtained measures shows that differences among the applications are best represented by differences in Lines of Code. It is likely that the actual size of an application that requires substantial amounts of data processing is not fully represented by functional size measures. In summary, the paper shows that not taking into account data processing dramatically limits the expressiveness of the size measures. Practitioners that use size measures for effort estimation should complement functional size measures with measures that quantify data processing, to get precise effort estimates.**

Keywords- functional size measurement; Function Point Analysis; IFPUG Function Points;COSMIC method.

I. INTRODUCTION

The most popular Functional Size Measurement (FSM) methods, i.e., IFPUG (International Function point User Group) [1][2][3] and COSMIC (Common Software Measurement International Consortium) [4]– adopt a concept of "functionality" that is based mainly on two elements:

- the processes, named Elementary Processes (EP) in IFPUG and Functional Processes (FPr) in COSMIC;
- the data that cross the boundary of the application being measured or are used (i.e., read or written) in the context of a process.

Quite noticeably, neither method satisfactorily considers the amount of data processing involved in a process. As a matter of fact, Function Point Analysis proposes an adjustment of the size based on the complexity of data processing, but, as discussed in Section VII, quite imprecisely and ineffectively, while the COSMIC method does not take the amount of data processing into account at all.

The goal of the paper is to provide evidence, by using an example, that not considering data processing dramatically limits the expressiveness of functional size measures.

The core of the paper can be described as follows:

- Two applications are specified. These applications are similar with respect to the aims and functionality offered to the user, but they are very different in the amount and complexity of the processing required.
- The two applications are modeled and measured according to the IFPUG and COSMIC rules.
- It is highlighted that the two applications have the same functional size measures, even though the amount of functionality to be coded in the two cases is enormously different.
- In fact, when measured via Lines of Code, it is apparent that the implementations of the two applications have quite different sizes. The reason is that more data processing clearly requires more code.

The conclusion is that using only the functional size to estimate development effort is likely to yield huge errors for complex applications. Since size measures are used for effort estimation, using functional size measures to size complex applications (i.e., programs that require a substantial amount of data processing) may lead to large (and dangerous) effort underestimations.

The paper is structured as follows. Section II reports a few basic concepts of functional size measurement. Section III illustrates the case studies used in the paper. Section IV describes the models and measures of the considered applications: the collected measures are then compared in Section V. Section VI discusses the alternatives that should be considered for complementing standards functional size measures with measures that represent data processing. Section VII accounts for related work. Finally, Section VIII draws conclusions and briefly sketches future work.

 ISBN: 978-1-61208-367-4

II. FSM CONCEPTS

Functional size measurement methods aim at providing a measure of the size of the functional specifications of a given software application.

Here, we do not need to explain in detail the principles upon which FSM methods are based. Instead, it is important for our purposes to consider what is actually measured, i.e., the model of software functional specifications that is used by the Function Point Analysis (FPA) and COSMIC methods.

The model used by FPA is given in Figure 1. Briefly, Logical files are the data processed by the application, and transactions are the operations available to users. The size measure in Function Points is computed as a weighted sum of the number of Logical files and Transactions. The weight of logical data files is computed based on the Record Elements Types (RET: subgroups of data belonging to a data file) and Data Element Types (DET: the elementary pieces of data). The weight of transactions is computed based on the Logical files involved –see the FTR (File Type Referenced) association in Figure 1– and the Data Element Types used for I/O.

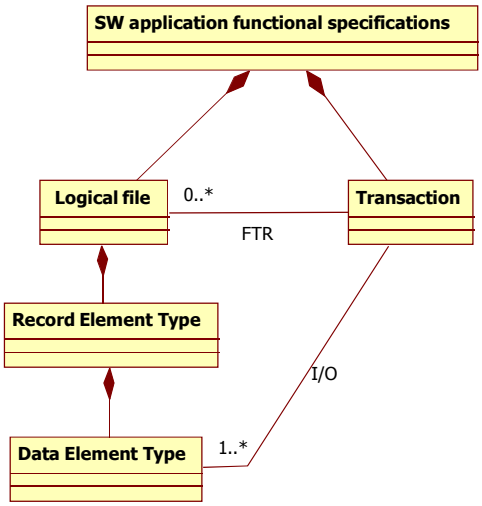


Figure 1. The model of software used in Function Point Analysis.

It is possible to see that in the FPA model of software, data processing is not represented at all.

The model used by COSMIC is given in Figure 2. The size of the functional specification expressed in COSMIC function points (CFP) is the sum of the sizes of functional processes; the size of each functional process is the number of distinct data movements it involves. A data movement concerns exactly one data group.

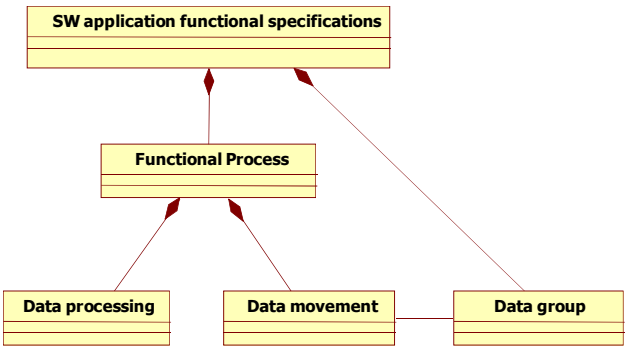


Figure 2. The model of software used by the COSMIC method.

Neither data groups nor data processing are directly used in the determination of an application's functional size. In particular, data processing is not measured at all. The COSMIC method assumes that a fixed amount of data processing is associated with every data movement; however, it is not so, in the examples considered in this paper.

III. CASE STUDIES

In this section, we describe the functional specifications of the two software applications that will be used to test the functional sizing ability of FPA and COSMIC.

The chosen applications are programs to play board games against the computer. They are similar with respect to the provided functionality, but require different amounts of data processing.

The specifications that apply to both applications are as follows:

- The program lets a human player play against the computer.
- The program features a graphical interface in which the game board is represented.
- The player makes his/her moves by clicking on the board. Illegal moves are detected and have no effect. As soon as the human player has made a move, the computer determines its move and shows it on the board.
- When the game ends, the result is shown, and the player is asked if he/she wants to play another game.

A. *A Software Application to Play Tic-tac-toe*

Tic-tac-toe is a very simple, universally known game. It is played on a 3×3 board, as shown in Figure 3. Each player in turn puts his/her symbol in a free cell. The first player to put three symbols in a row (horizontally, vertically or diagonally) wins. When the board is filled and no three-symbol row exists, the match is tie.

 ISBN: 978-1-61208-367-4

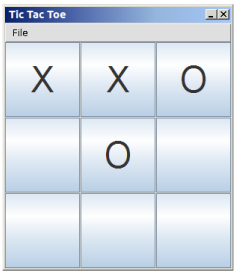


Figure 3. Tic Tac Toe playing board.

Playing Tic-tac-toe is very simple. In fact, to play optimally, a software program has just to evaluate the applicability of the following sequence of rules: the first applicable rule determines the move:

1) If there is a row such that two cells contain your symbol, and the third cell X is empty, put your symbol in the free cell X.
2) If you are the first to move and this is your first move, put your symbol in the central cell.
3) If there is a row in which your opponent has two symbols and the third cell X is free, put your symbol in the free cell X.
4) If there is a free cell X such that putting your symbol there results in two rows, each one having two cells occupied by your symbol and the third cell free, put your symbol in cell X.
5) If there is a row in which you have one symbol and the other two cells X and Y are free, put your symbol in cell X or in cell Y.

The code that implements the playing logic described above is very simple and very small: we can expect that a few tens of lines of code are sufficient to code the game logic.

B. *A Software Application to Play "five in a row"*

Five in a row (aka Gomoku) can be seen as a generalization of Tic-tac-toe. In fact, it is played on a larger board (typically 19×19, as in Figure 4) and the aim of the game is to put five symbols of a player in a row (horizontally, vertically or diagonally).

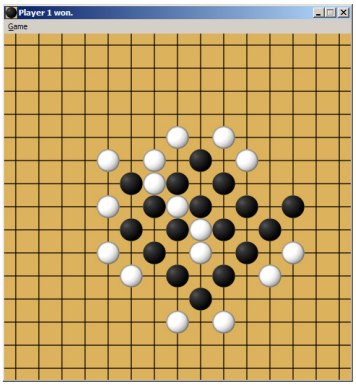


Figure 4. Gomoku playing board.

The functional specifications of Gomoku are exactly the same as the specifications of Tic-tac-toe, except that

a) The size of the board is larger
b) The number of symbols to put in a row is 5 instead of 3.

The combinations of symbols and free cells that can occur in a Gomoku game are many more than in a Tic-tac-toe game. Accordingly, a winning strategy is much more complex, as it involves considering a bigger graph of possibilities.

As a matter of fact, Gomoku has been a widely researched artificial intelligence research domain, and there are Gomoku professional players and tournaments.

Accordingly, we can safely state that Gomoku is a much more complex game than Tic-tac-toe, and it requires a huge amount of processing, so that the machine can play at a level that is comparable with that of a human player.

On the contrary, Tic-tac-toe is a very simple game: you do not need to be particularly smart to master it and always play perfectly.

IV. APPLICATION SIZING

A. *A Software Application to Play Tic-tac-toe*

Let us measure the Tic-tac-toe specifications given in Section III.A above, starting with IFPUG Function Points.

The software model to be used involves just a Logical data file: the board and a matrix of cells, each having one of three possible values (circle, cross, free).

The software model to be used involves the following elementary processes:

– Start a new game.
– Make a move.

It is not necessary to consider details (RET, DET) to see that the Logical data files is a simple Internal Logical File (ILF), contributing 7 FP.

Similarly, it is not necessary to consider details (FTR, DET) to see that:

– Start a new game is a simple External Input (EI), contributing 3 FP.
– Make a move is a simple external output, contributing 4 FP. One could wonder if this operation should be considered an input (because the move involve inputting a position) or an output (because of the computation and visualization of the move by the computer). We consider that the latter is the main purpose of this transaction, which is thus an external output.

In summary, the FPA size of the Tic-tac-toe application is 14 FP.

The COSMIC functional processes of the application are the same as the FPA elementary processes. When measuring the application using the COSMIC method, we have to consider the data movements associated with each functional process:

– Start a new game involves clearing the board and possibly updating it, if the computer is the first to move (a Write) and showing it (a Read and an Exit). Therefore, this functional process contributes 3 CFP.
– Make a move involves entering a move (an Entry), updating the board with the human player move (a

Write), reading it (a Read), and then updating it again with the computer move and showing it (an Exit). In addition, if a move concludes the game, the result is shown (an Exit). Therefore, this functional process contributes 5 CFP.

In summary, the COSMIC size of the Tic-tac-toe application is 8 CFP.

Since we are also interested in indications concerning the amount of computation performed by the application, we selected an open source implementation of Tic-tac-toe and measured it.

To evaluate the "physical" size of the Tic-tac-toe application, we looked for an open source application that implements the specifications described above. One such application is the program available from [8].

The main measures that characterize the code are given in TABLE I.

TABLE I. MEASURES OF THE TIC-TAC-TOE APPLICATION CODE

Measures	Tic-tac-toe [8]	
	Total	***AI part***
LoC	172 (118 statements)	66 (52 statements)
McCabe	3.6	5
Num. classes	2	1
Num. methods	17	7

In TABLE I (and in TABLE II), column "AI part" indicates the measures concerning exclusively the part of the code that contains the determination of the computer move.

In the LoC line, we reported both the number of lines and the number of actual statements. The latter is a more precise indication of the amount of source code. We also reported the mean value of McCabe complexity of methods.

B. A Software Application to Play "five in a row"

The functional size measures of the Gomoku application are exactly the same as the measures of the Tic-tac-toe application. In fact, the specifications of the two applications are equal, except for the board size and winning row size, which do not affect the measurement, because both IFPUG FPA and COSMIC consider data types, not the value or number of instances.

As for Tic-tac-toe, we selected an open source implementation of Gomoku and measured it. More precisely, to take into account that a programmer may aim at developing a program capable of more or less sophisticated "reasoning," we considered a few different implementations of Gomoku.

In this case, to evaluate the "physical" size of the application, we also looked for an open source application implementing the specifications described above. One such application is the Gomoku application available from [9].

The main measures that characterize the code are given in TABLE II.

TABLE II. MEASURES OF THE GOMOKU APPLICATION CODE

Measures	Gomoku [9]	
	Total	***AI part***
LoC	832 (395 statements)	425 (234 statements)
McCabe	3	5.95
Num. classes	12	3
Num. methods	63	21

Measures in TABLE II were derived using the same tools and have the same meaning as the measures in TABLE I.

V. COMPARISON OF MEASURES

The measures reported in the previous section show that we can have two applications that have the same functional size, but very different code size (the Gomoku applications are over four times as big as the Tic-tac-toe application). Considering the nature of these applications, the difference in code is largely explained by the different amount of processing required. In the case of Tic-tac-toe, the number of possible moves is very small, as is the number of different possible configurations that can be achieved by means of a move: hence, every move computation has to explore a very small space. The contrary is true for the Gomoku application. The consequence is that Gomoku requires an amount of code devoted to move computation that is over 6 times the code required by Tic-tac-toe (or 4.5 times, if we consider the number of statements instead of LoC).

These observations suggest two important considerations:

1. The definitions of Function Point Analysis and the COSMIC method do not properly take into account the amount of processing required by software functional specifications.
2. If we assume –as is generally accepted– that the effort required to implement a software application is related to the number of Lines of Code to be written, the possibility of having widely different sizes in LoC for applications that have the same functional size means that functional size is not a good enough predictor of development effort.

The observation reported at point 2 above does not apply only to the coding phase. The difference in the number of classes and methods suggest that also the effort required by design and testing activities is better estimated based on measures that represent the size of the code structure –like the number of classes– rather than the functional size.

As a final remark, we can observe that McCabe complexity is similar for the two considered applications. This means that Gomoku does not need more complex code, but just more code. In other words, it is the difference in the amount of data processing, not in the complexity of the processing that is relevant, and that existing functional size fail to represent.

VI. DISCUSSION: WHAT SOLUTIONS ARE POSSIBLE?

The usefulness of the evidence given in this paper stems from a few well-known facts:

 ISBN: 978-1-61208-367-4

- We need to estimate, during the early phases of a project, the overall software development effort.
- Development effort has been widely reported to be directly related to the size in LoC of software. Unfortunately, the size in LoC is not available in the early phases of projects, when estimates are most needed.
- Therefore, we need FSM methods, i.e., we need measures of functional specifications, because specifications are available in the early phases of projects.
- In this paper, we provide some evidence that current FSM methods appear limited in representing the amount of data processing required by functional specifications. Therefore, we need to somehow enhance FSM methods to remove such limitation.

So, we are facing the following research question: how can we improve FSM methods so that the delivered functional size measures account for the amount of data processing described or implied by the functional specifications?

This is an open research question. Providing a final answer to it can be achieved after a substantial amount of further studies. In the following sections, we report a few observations, ideas and evaluations that could be useful considering when tackling the problem.

A. Software Models

FSM methods –like any measurement method– are applied to models of the object to be measured. Hence, a rather straightforward consideration is that data processing must be represented in the model that describes the software application to be measured.

We can observe that the conceptual model of software proposed in the COSMIC method includes data processing, but no criteria or procedures for measuring data processing are given in the context of the COSMIC method.

In COSMIC, data processing is a sub-process of a functional process. Therefore, functional processes should be described in a manner that makes it possible to identify and measure the extent of data processing that occurs within a functional process.

Given the similarity of COSMIC functional processes and FPA elementary processes (or transactions) any technique used to enhance the expressivity of COSMIC models as far as data processing is concerned should be readily applicable to FPA models as well.

B. Software Specifications

A question that should be considered is if the information required for identifying and measuring data processing is always available from the software specifications that are derived from user requirements.

Functional Size Measurement methods use models of functional specifications: if functional specifications do not include information on data processing, neither will their models, and FSM methods will not be able to account for data processing.

So, another open question is the following: is it necessary to go beyond user requirements related specifications to be able to represent data processing? In other words: should elements of design be anticipated, to get better measures of the amount of data processing to be implemented?

C. Qualitative Knowledge

Current FSM methods are inherently quantitative. Even if some measurement activities –like deciding if two sets of data should be two RET of a unique logic file or they should belong to separate logic files– involve some subjectivity, they are always meant to provide measures (the number of ILF, RET, etc.) according to ratio scales.

One could wonder if the use of more qualitative knowledge, derived through inherently subjective evaluations and expressed via ordinal scales, would more suitable for expressing the relevant information concerning data processing.

For instance, after talking with stakeholders, an analyst could easily classify the functional process “Make a move” of the Tic-tac-toe application as very simple, while the same process of the Gomoku application could be classified as very complex.

D. Towards a Measure of Data Processing

As mentioned above, proposing a solution to the problem outlined above is very difficult. Here we outline a couple of directions to be considered when addressing the problem.

A first consideration concerns the level of description of data processing. At a high level, the complexity of the processes in terms of number of different cases to be considered could easily determine the amount of data processing required. Consider for instance a process that starts by identifying users: if the specifications indicate that the user can be identified in three different ways (e.g., by name, by social security number, and by email address) it is likely that it will have to process three times as much data as a process that identifies users in a single way.

Another observation concerns how to differentiate functionalities. A possibility is to account for the internal states a function has to deal with. In the case of tic-tac-toe, the number of states in which the game can be is quite small; on the contrary, the states of a Gomoku game are very numerous. Accordingly, the amount of computation could be proportional to the number of states, since the function has to properly deal with all states. However, the quantification of data processing could be further complicated by the presence of equivalent states, i.e., sets of states that are managed in the same way, so that having N or N+1 states in such sets would not affect the amount of processing required. For instance, a data increase function has to account for months having 28, 30, or 31 days: the fact that there are 7 months having 31 days and just one having 28 days is irrelevant.

VII. Related Work

Although several FSM methods (e.g., Mark II FP, NESMA and FiSMA) have been proposed as extensions or replacements of Function Point Analysis, very little attention has been given to the measurement of data processing.

 ISBN: 978-1-61208-367-4

Function Point Analysis and other methods –like Use Case Points [5]– introduce a mechanism for "adjusting" the size measure to take into account additional complexity factors that are likely to increase the effort required for implementation. In fact, among FPA value adjustment factors (VAF) we find "Complex processing," which represents to what degree the application includes extensive logical or mathematical processing. This mechanism is similar to what we need, but has a few shortcomings, including:

- In FPA the considered VAF's value increases the application size by 5%: two orders of magnitude less than needed in the Tic-tac-toe vs. Gomoku case.
- The VAF applies to the whole application, so that it is not possible to distinguish simple and complex processes.

The measure of Path [6][7] represents the complexity of processes in terms of the number of execution paths that are required for each process. Although this measure proved fairly effective in improving effort estimation based on functional size measures, it is not applicable in cases like those considered in this paper, since the alternative courses of the specified processes are not known.

VIII. CONCLUSIONS

In this paper, we have shown by means of examples that functional size measurement methods fail to represent the amount of data processing required by software functional specifications.

Since we discussed just one example, one could wonder how general are the results reported in the paper. As to this issue, it is easy to see that the limits of FSM discussed in the paper apply to several programs. Consider for instance software measurement programs: from the point of view of functional size, all the measurement functions that read a set of source files and deliver a numeric value are equivalent. However, it is clear that measuring LoC is easier (i.e., it involves less data processing) than computing McCabe complexity, which in its turn is easier to compute than most coupling measures.

The work reported in the paper indicates that we need a measure that can complement Function Points or COSMIC Function Points to represent the amount of data processing that is required to provide the required functionality.

We are interested to represent and quantify the amount of data processing not because of an abstract interest in the definition of functional size measures, but because –as shown in the paper– data processing is logically related to code size, which is known to determine the amount of development effort required to build a software application.

How to measure the amount of data processing required by the specifications of a software application is an open research question of great practical interest that should receive much more attention than it currently does.

ACKNOWLEDGMENT

The work presented here has been partly supported by the FP7 Collaborative Project S-CASE (Grant Agreement No 610717), funded by the European Commission and by project "Metodi, tecniche e strumenti per l'analisi, l'implementazione e la valutazione di sistemi software," funded by the Università degli Studi dell'Insubria.

REFERENCES

[1] A. J. Albrecht, "Measuring Application Development Productivity", Joint SHARE/ GUIDE/IBM Application Development Symposium, 1979, pp. 83-92.

[2] International Function Point Users Group. Function Point Counting Practices Manual - Release 4.3.1, January 2010.

[3] ISO/IEC 20926: 2003, Software engineering – IFPUG 4.1 Unadjusted functional size measurement method – Counting Practices Manual, Geneva: ISO, 2003.

[4] COSMIC – Common Software Measurement International Consortium, The COSMIC Functional Size Measurement Method - version 3.0.1 Measurement Manual, May 2009.

[5] G. Karner, "Resource estimation for objectory projects". Objective Systems SF AB, 17. 1993.

[6] G. Robiolo and R. Orosco, "Employing use cases to early estimate effort with simpler metrics". Innovations in Systems and Software Engineering, 4(1), 2008, pp. 31-43.

[7] L. Lavazza and G. Robiolo, "Introducing the evaluation of complexity in functional size measurement: a UML-based approach". ACM-IEEE Int. Symp. on Empirical Software Engineering and Measurement, September 2010.

[8] http://algojava.blogspot.it/2012/05/tic-tac-toe-game-swingjava.html.

[9] https://github.com/whsieh/gomoku

 ISBN: 978-1-61208-367-4

An Exploration of the Application of Usability Evaluation Methods by Disabled Users

Khalid Al-Nafjan
Software Engineering Department
King Saud University
Riyadh, Saudi Arabia
kalnafjan@ksu.edu.sa

Mona Al-Zuhair
Software Engineering Department
King Saud University
Riyadh, Saudi Arabia
433203603@student.ksu.edu.sa

Layla Al-Salhie
Healthcare IT Affairs
King Faisal Specialist Hospital &RC
Riyadh, Saudi Arabia
lalsalehi@kfshrc.edu.sa

***Abstract*— The involvement of system users during the system usability evaluation with the full awareness of their specific nature and characteristics is a key factor for achieving effective usability evaluation results. However, conducting usability evaluations for systems designed for disabled users is a challenging process that requires further considerations as compared to other ordinary usability evaluation procedures. This is due to the special needs and conditions of disabled users that must be considered while performing the usability evaluation. Therefore, it is essential to assess the effectiveness of different usability evaluation methods to help the evaluator selecting the most suitable ones for a particular system and a particular user group. The main contribution of this paper is to conduct an analysis of the effectiveness of applying several usability evaluation methods by disabled users. This analysis is based on the special characteristics of users with disabilities and on what adjustments should take place before the evaluation process begins. After conducting this exploratory analysis, we found that usability evaluation methods including inspection and testing methods can be applied to special needs users but many considerations should take place before selecting which methods are most effective. We believe that this work is particularly useful for the novice designers and usability engineers who have never conducted usability evaluations by disabled users before.**

Keywords-UEM; Disabled users; Usability evaluation.

I. INTRODUCTION

There is a global agreement on that the usability is a key aspect of a software product's success. System usability can be viewed as the studies conducted that aim to answer the question of whether the system is good enough to satisfy the user's needs [2]. In order to properly acquire the desired level of usability in a software system, a disciplined approach should be followed. For that, usability engineering concept has emerged into software engineering to represent this disciplined approach [14]. Several usability engineering process models exist in literature that share an essential activity which is the usability evaluation [3]. Usability evaluation is an iterative process that encompasses a continuous measurement of the system's current usability level; this process continues to repeat until the desired usability level is reached.

In literature, several techniques, methods and guidelines exist that shape the usability evaluation activity. A usability evaluation method (UEM) is a process for producing a measurement of usability: in evaluation, there is an object being evaluated and a process through which one or more attributes are judged or given a value [18]. The standard output for all UEMs is a list of the potential usability problems [7]. These UEMs can be classified in several ways; a common way is to classify them into empirical user testing methods and usability inspection methods, according to the user involvement. While the user testing category covers methods that involve representative users as participants, the usability inspection category, on the other hand, includes methods that can be applied without user involvement [16]. User testing also includes developing realistic scenarios of the tasks that the users are required to perform [17]. Assessing the effectiveness of different UEMs is essential to help the evaluator selecting the most suitable UEMs for a particular system and a particular user group. This effectiveness is related to several factors such as the type of the systems, the nature and time of the usability study among the development lifecycle, the characteristics of test participants, funding and other facilities [2]. Several measures can be used for assessing UEMs effectiveness such as: the ratio and severity of usability problems detected, the ratio of task success and number of comments elicited [6].

The main contribution of this paper is to conduct an analysis study of the effectiveness of applying several usability evaluation methods by disabled users based on the special characteristics of such users and the adjustments that should take place before the evaluation process begins. However, the study was based on analyzing the literature and reviewing the fields that focus on the application of different UEMs with different disabilities. And the results that we obtained during this study were based on our findings and experience after analyzing these fields. The following UEMs will be analyzed in this paper in regard to their application by disabled users: Inspection methods, thinking aloud, attention analysis, field observation, coaching method, questionnaires and interviews. The rest of the paper is organized as follow: Section 2 presents the related works that studied the application of different UEMs by disabled users. Section 3 provides an exploratory review of applying different UEMs for users with specific disabilities along with the resultant considerations. Section 4 presents

 ISBN: 978-1-61208-367-4

summarization of the analysis results. In Section 5, we concluded the paper with a summary and the expected future work.

II. RELATED WORK

Most of the related work had discussed – as a part of a system development process- the results of conducting a specific UEM for evaluating the system's usability by users with a specific type of disability. However, few works discussed the effectiveness of applying different UEMs by different disabled users and highlighted the challenges faced.

Regarding the challenges of the recruitment of special – non regular- test users, Brush et al. [4] discussed the problem of the availability of user representatives. They found it difficult to find sufficient professional users of testing the usability of an urban planning tool locally because users were geographically distributed. They conducted both local and synchronous remote usability testing and found the results comparable. The effectiveness of applying remote evaluation by disabled users was evaluated by Petrie et al. [9]. They presented two usability evaluation studies with disabled users. One was remotely conducted and in asynchronous way while the other was ordinary local evaluation. In the remote evaluation, there were two cases; summative evaluation and formative one. The resulting quantitative data of the local and remote cases were comparable. However, there was a difference in the data amount and richness in the favor of local evaluation.

Regarding usability evaluation by slow learning users, Abdollah et al. [1] developed a multimedia courseware learning tool for slow learners and performed a usability evaluation of the tool by the slow learners along with heuristic evaluation with teachers and one parent. Evaluation results showed that users with this disability were able to participate in the "efficiency" and "easy to learn" measurement testing while they were unable to participate in the "satisfaction" measurements testing considering their lack of respond abilities to written questionnaires. As for the deafness disability, Roberts and Fels [5] provided two studies that proved the viability of using the Think Aloud Protocol (TAP) method as a UEM in collecting gestures from deaf sign language users. Their study showed a similar success rate of using gestural TAP for deaf people and verbal TAP for hearing people. As for applying UEMs by blind users, Chandrashekar and Fels [8] assessed the applicability of conventional TAP method to blind users who uses a screen reader to access websites. They found that TAP cannot be used by such users in its popular form as a concurrent verbal protocol; it will instead require adjustment to be useful for blind users. However, the best approach for TAP adjustment wasn't determined in the study.

Regarding the usability evaluation by users with cognitive disabilities, Lepistö and Ovaska [20] performed a think aloud usability test and found that it didn't work well with this user group. They also conducted an informal walkthrough and found that this method showed effectively which parts of the application interest the participants most. Their study concluded that for evaluating usability by users with cognitive disabilities, several complementary methods might be needed to collect data, and these methods should be adjusted to suit the characteristics of such participants. Another conclusion is that without the observational methods, many usability problems would have been missed. Authors also emphasized that evaluators should focus on gaining the participants' trust before the evaluation sessions.

Rømen and Svanæs [19] have studied the usability evaluation by users with several disabilities (blind, weak-sighted, motor impaired and dyslexic) as a part of their validation of the usefulness of using Web Content Accessibility Guidelines (WCAG) as a heuristic for website accessibility. Several techniques were used for the testing process: a "think aloud" was conducted at first; then a short interview was conducted after completing the test tasks in order to uncover further problems. Evaluators also used a mobile usability lab which allowed the disabled users to be tested at their workplace and home using their own computer and technical aides. The study results showed that only 27% of the identified website accessibility problems could have been identified through the use of WCAG heuristics.

The works reviewed above have addressed the application of specific types of UEMs by disabled users as part of presenting the development process of a software system. However, this paper contributes to explore the effectiveness of applying number of UEMs (Inspection methods, thinking aloud, attention analysis, field observation, coaching method, questionnaires and interviews) for users with disabilities and to present the related conditions and considerations that would customize these UEMs to fit a specific disability. Furthermore, this contribution has been conducted by analyzing and reviewing the literature and the fields that focus on the application of different UEMs with different disabilities; and the obtained recommendations and considerations were based on our findings after analyzing these fields. We think that this work will help novice designers and usability engineers by giving them deeper insight on the areas that they should consider during the usability evaluation for systems designed for disabled users.

III. EXPLORATORY ANALYSIS OF APPLYING USABILITY EVALUATION METHODS BY USERS WITH DISABILITIES

Based on the nature of systems designed for users with disabilities and based on the disabled users' characteristics, the most effective UEMs can be selected in order to discover all the possible usability issues that impact the system's users [10]. In this study, we analyzed and discussed the application of different UEMs by users with different disabilities focusing on the users' special characteristics. The analysis study was based on reviewing the literature and the fields and the obtained result was based on our findings and experience after performing this analysis study. However, the result was a set of recommendations and considerations that should take place before conducting usability evaluation by disabled users.

A. Inspection Methods and Disabled-Users

Before testing the system by real disabled users, inspection methods such as heuristic evaluation, cognitive

walkthrough and action analysis can be applied in order to find and resolve the general and common usability issues based on the expertise of usability and design. The system then becomes ready for testing by disabled users. Although there are many guidelines for designing accessible systems for users with different kinds of disabilities, usability engineers and the designers lack the experience with disabled people characteristics and their different assistive technologies. Therefore, many of the system's accessibility criteria are missed. In this case, the involvement of specialized therapists during the analysis becomes important in order to improve the effectiveness of the inspection methods in order to find more disability-specific usability issues. For example, in analyzing system designed specifically for users with physical impairment; the physical therapist can be participated in order to define the specific limitations that such users may face during the interaction with the system. And when we speak about system designed specifically for children with disability, like for example slow learning children, the involvement of persons like parents or teachers who are interacting with the child very closely and aware with most of the issues that this kind of children have, this can add more value to the inspection method and it will also help finding more usability issues. Generally we can say that relaying only on these inspection methods can find some general usability problems, but in order to find more detailed problems and useful information on how to improve the usability, it is necessary to conduct the Usability Testing Methods by actual disabled users [9].

B. Test Methods and Disabled-Users

Usability Testing methods are conducted by real system users and their main objective is to identify problems that users face when dealing with the system. These tests provide precise identification and description of the usability issues that may lead to system re-designing [11]. In particular, for disabled users, many considerations should take place before conducting the test and some of these considerations are common for all type of disabilities; while others are specific for certain disability. Generally, for all kinds of disabilities, the testing environment (either room or laboratory) should be prepared and organized for the disabled user. For example, in case of physical impairment users, testing place should allow enough area for a wheelchair to get in, move around and face the computer. Furthermore, an important thing to consider before conducting tests by disabled users is the different profiles within this user group, i.e. disabled user may be employed or unemployed; educated or under-educated; technology 'power users' or computer illiterate. Information about these differences should be gathered in order to deepen the obtained results. Moreover, it is important to select what system interfaces to be tested by the disabled users. Therefore, different test tasks can be prioritized based on their importance in the system and also based on the amount of user interaction involved. This is because the areas of a system that have the most usability problems are the ones involving the most user interaction [17].

There are many usability testing techniques such as: Thinking aloud, Attention Analysis, Field Observation, Questionnaires and others. Here, we will go through some of these techniques and analyze their effectiveness when being conducted by users of specific disabilities:

- Thinking aloud

The Thinking aloud method requires the user to verbalize all his/her cognitions when interacting with the system. It is considered one of the most effective techniques in identifying usability problems [11]. When conducting this method by disabled users, the evaluator has to consider the participants' disability before starting the test. For blind and visually impaired participants, they usually use a screen reader as their main assistive technology. However, the evaluator should focus on both the screen reader and the participant voice and expressions. And in this case, he/she has to position the audio recording equipment close enough to hear the screen reader. The evaluator can also use separate audio equipment for the participant's voice and for the screen reader, that way, when analyzing the data, the evaluator can combine between the two recordings [12]. Another case of disabled users are the deaf participants. In this case the evaluator has to record both the participant and the interpreter voice, depending on the situation. If the participant doesn't speak at all, the evaluator can record only the interpreter voice. If the participant speaks some, the evaluator probably want to record both of them. This technique will obviously put more stress on the participant because it is unnatural to his/her to express his thinking loudly and share it with others [11]. Therefore, we might say that think aloud method considered time-consuming and hard to apply for deaf participants since they have to share their thoughts with the interpreter who will give the answers to the evaluator. Also, Roberts and Fels [5] proved the viability of using this method in collecting gestural protocols from the sign language of deaf users and extract relevant usability issues and remarks.

- Attention Analysis

Attention Analysis method includes two categories: Attention-tracking and eye-tracking. In the attention tracking, the participant is asked to click on the areas in the system interface that he/she finds most noticeable. The eye-tracking method in the other hand requires special equipment in order to capture the users' eyes movement so the evaluator can analyze it and gain useful information on the noticeable interface elements. As we can see, this technique is inapplicable for blind participants. While for other disabilities it can be helpful in finding and analyzing which elements of the system are most distracting and how long users remain in certain sections of the system. Furthermore, this technique can be used for evaluating systems designed for aiding children with Autism syndrome by examine and identify the types of interface elements (i.e., animation) that attract and retain child's attention.

 ISBN: 978-1-61208-367-4

- Field Observation

This method involve an investigator who is observing the users as they work in their work environment, and taking notes on the activity that takes place there. Observation may be either direct where the investigator is actually present on the observation area or indirect where the task is recorded using a video recorder and later on the investigator can analyze it [15]. Allowing the observer to view what users actually do in their context, especially in case of disabled users, will add great value to the process of finding usability issues. Direct observation allows the investigator to focus attention on specific areas of interest and it will let him/her see how the disabled users use their assistive technologies, and which kind of daily practice they perform. And due to the nature of these users and their sensitivity to any new passive presence in their environment, the investigator should make sure that users are aware with the purpose of his/her presence and the main reasons of the observation. This is particularly important for mentally impaired and blind users who may be disturbed by a passive presence that they are not sure about.

- Coaching Method

In this technique the evaluator serves as the coach, where participants are allowed to ask any questions to an expert coach who will answer to the best of his/her ability. The purpose is to discover the information needs of users and find out the limitations in the system design to possibly redesign the interface to avoid the need for the questions [2]. This technique would help in case of blind participants who most of the times need guidance in order to make sure that they are in the right direction. Furthermore, it would help in case of children with learning difficulties since they need continuous help; and the coach can respond to their questions and give them the needed assistance.

- Questionnaires

Questionnaires are designed to help the evaluator in obtaining data about the users' subject judgment of the system and reflecting their level of satisfaction. It can be used to evaluate entire system or only partial aspects of the system. This technique is applicable for all kinds of disabilities, except disabled children and slow learners [1]. However, it can be performed in a much simpler form which is the "Interview" [13] in order to simplify the technique for disabled users. The issue with this technique is that it needs enough number of participants, not less than 30, in order to make sure that enough opinions have been collected [13]. However, establishing cooperative relationships with organizations of disabled people may help in recruiting the required number of user participants [9].

C. Performing the Test Methods Locally or Remotely

Finding and recruiting test users with special characteristics or specific demographics is a challenging task [9]. For example: it is not easy to find blind or deaf person who can be participated and committed to the system usability evaluation. This is obviously due to the special issues these users might have such as: transportation issues and the need for continuous assistance. As a result, finding test participants with disabilities is a problematic. Since their involvement is a key factor in usability tests, conducting the usability test remotely in the disabled user's own environment would be a good practice. Disabled users usually use assistive software or hardware technologies such as: screen magnification programs for partially sighted people, single handed keyboards [9]. They also configure these technologies, in a way that fits their needs. Therefore, having remote evaluations which involve performing the test in the users' areas is valuable, especially that the evaluator will be able to closely see how an assistive technology is being used by the disabled user and how this technology affects the usability of the system under testing. As a result, more detailed usability problems can be discovered. As mentioned by Petrie et al. [9], there are number of "remote evaluation techniques" such as: portable evaluation, local evaluation at remote site, remote questionnaire/survey, video conferencing and others. Each one of these techniques can be selected according to the users and the evaluators' conditions. In contrast, going to each individual participants and perform the test in their environments is considered costly and time-consuming; especially when we are talking about large scale projects that need number of participants with different disabilities who may be located in different areas. Therefore, conducting the test locally in a usability laboratory by having the participants attend the test place is more effective and it can save time, cost and effort. As per what has been mentioned above, where to conduct the usability test either locally or remotely is an important dimension that should be considered to obtain effective usability evaluation by disabled users.

D. Participants Independence

One of the important issues that affect the effectiveness of the usability test results is the participants' independence and the amount of their contact with the evaluator during the test. In most of the cases, disabled users need some guidance during the test from the evaluators. Such communication should be very well planned and organized in a way that will not affect the accuracy and validity of the test results, since intensive communication can distract the participants' attention and prevent the evaluators from getting reliable results. Meanwhile, lack of help and guidance during the test could lead the disabled user to struggle in one task or become in the wrong direction. In case of deaf users who speech-reads, the evaluator should sit in a position in front of the test participant to allow him/her to read the evaluator's lip and face expressions during the communication, this could be distracted to the user if it is not kept to the minimum during the test. As for slow

 ISBN: 978-1-61208-367-4

learning children, communication has to be through the learning difficulties' specialist who knows how to interact with children and provide the appropriate assistance.

E. Synchronous or Asynchronous Tests

Selecting among synchronous or asynchronous tests is about assessing the need to have the evaluator and the participant performing the usability test at the same time. In Asynchronous test, the evaluator can provide the participant the test details and manuals and leave him, and after finishing the test, the evaluator can collect the results data like video recording or screen recording tape. Synchronous tests in the other hand, implies having the evaluator participates with the user along the test time by observing him/her while performing the tasks. This way, the evaluator can explore more information like the non-verbal behavior of the participant can reveal more usability problems and their causes. In addition, one of the most important benefits of synchronous tests is that the evaluator will directly and carefully observe the disabled user and see how he/she interact with the system using assistive technology. This will ensure a deep understanding of different usability issues that should be considered in system design [9][11].

IV. SUMMARIZATION OF THE ANALYSIS RESULTS

Generally, we can say that when evaluating the usability for systems designed for disabled users, combination between UEMs can be performed in order to find and discover most of the possible usability issues. Meanwhile, selecting the most effective UEMs should be done under many considerations like: the system goals, users' disability type and the project's time and cost constraints. Based on the previous section, results of analyzing the effectiveness of UEMs by disabled users are summarized in Table1. These results are depicted in the shape of considerations and recommendations along with the justifications behind the selection of these recommendations.

TABLE I. ANALYSIS RESULTS OF THE EFFECTIVENESS OF UEMS BY DISABLED USERS

Disability	Considerations/ Recommendations	Justification
Complete - Partial Deaf	Involve sign language interpreter in the test.	For effective communication with deaf users.
	Plan and manage the interaction during the test.	To avoid any distraction during the test for speech-reads users in order to get accurate reliable test results.
	Combination of test methods: thinking aloud and questionnaire is recommended to apply.	As per the review of previous works, these methods have shown effective results.
Complete -Partial visual impairment	Consider the screen reader assistive tool during the test.	Visually-impaired users rely heavily on the screen reader. therefore to get accurate results, these tools should be considered during the test.
	Provide required assistance when needed.	Based on the special nature and the difficulties faced by such users when using the different systems, immediate and direct assistance should be provided during the test.
	Omit the "task completion time" constraint.	Visually-impaired users spend more time on performing tasks than other disabilities. Therefore, the time constraint should be removed during the test. This ensures more flexibility that helps them complete the tasks.
	Use automatic validation tools.	These users use assistive technology heavily. Therefore, checking the compatibility of the developed system and the assistive technologies using these tools is important.
	Combination of inspection methods is recommended: heuristic evaluation and test methods (coaching method).	As per the review of previous works, these methods have shown effective results by visual impairment users.
Physical impairment	Perform synchronous remote test (in the user environment).	Due to the different obstacles these users may face to attend usability test lab, performing remote test allows more of them to participate in the test.
	Involve physical therapists for effective inspection method.	Such therapists can define the impacts on such disabled users and their limitations during system interaction.
	Combination of inspection methods is recommended: heuristic and test methods (field observation, questionnaire or interview).	As per the review of previous works, these methods have shown effective results by such users.
Children with disabilities: Slow learning, Autism	Involve parents and learning difficulties' specialist for effective inspection methods.	Involving them can reveal more usability issues and assist in communication.
	Provide clear simple guidance and instruction.	Based their special nature and the difficulties they face when using computer applications, it is important to provide users with clear and simple guidance.
	Combination of inspection methods is recommend: heuristic evaluation and test methods (attention analysis, coaching method)	As per the review of previous works, these methods have shown effective results by such users.
Cognitive and Mental disabilities	Involve cognitive and mental health specialist for effective inspection methods.	Such involvement can reveal more usability issues and enhance the communication omng the test.
	Provide clear and simple guidance and instruction before the test.	Based on their special nature and the difficulties they face when using computer applications, it is important to provide users with clear and simple guidance.
	Combination of inspection methods is recommended to apply: informal walkthrough and test methods (field observation).	As per the review of previous works, these methods have shown effective results by such users
	The observer or test facilitator should build good relationship with the test participants and gain their trust before the test.	Due to the special nature of these users and their sensitivity, it is important to gain their trust to facilitate the communication during the test.

 ISBN: 978-1-61208-367-4

V. CONCLUSION AND FUTURE WORK

In this study, we analyzed and discussed the application of different UEMs by users with different disabilities focusing more on the users' special characteristics. The goal was to explore the effectiveness of applying number of UEMs with users with disabilities; and to present the related conditions and considerations that would customize these UEMs to fit a specific disability. However, the study was based on analyzing the literature and reviewing the fields that focus on the application of different UEMs with different disabilities. And the results that we obtained during this study were based on our findings and experience after analyzing these fields. We think that this work will help novice designers and usability engineers who have no prior experience with conducting usability evaluation with disabled users. However, this work will give them deeper insight on specific areas that they should consider during the evaluation. The future work will be conducting empirical evaluation with real disabled users in order to assess the effectiveness and accuracy of the obtained recommendations and results during this study.

REFERENCES

[1] N. Abdollah, W.F.W. Ahmad, and E.A.P. Akhir, "Development and usability study of multimedia courseware for slow learners: 'Komputer Saya'," Computer & Information Science (ICCIS), 2012 Int. Conf. on, vol. 2, June 2012, pp. 1110-1114, doi:10.1109/ICCISci.2012.6297192.

[2] J. Nielsen, Usability Engineering, Morgan Kaufmann Publishers Inc., San Francisco, 1993.

[3] B. Thurnher, "Usability engineering," Course No.: 188.314. Quality Software Engineering (QSE) Research Group. Institute of Software Technology and Interactive Systems (IFS), Sept. 2004.

[4] A J. Brush, M. Ames, and J. Davis, "A comparison of synchronous remote and local usability studies for an expert interface," In CHI '04 Extended Abstracts on Human Factors in Computing Systems . ACM, New York, NY, USA, 2004, pp. 1179-1182, doi:10.1145/985921.986018.

[5] V. Roberts and D. Fels, "Methods for inclusion: employing Think Aloud Protocols in software usability studies with individuals who are deaf," Int. J. Hum.-Comput. Stud. 64, June 2006, pp. 489-501, doi:10.1016/j.ijhcs.2005.11.001.

[6] A. Fernandez, S. Abrahao, and E. Insfran, "A systematic review on the effectiveness of web usability evaluation methods," Evaluation & Assessment in Software Engineering (EASE 2012), 16th Int. Conf. on, May 2012, pp. 52-56, doi:10.1049/ic.2012.0007.

[7] H. R. Hartson, T. S. Andre, and R. C. Williges, "Criteria for evaluating usability evaluation methods," Int. Journal of Human-Computer Interaction, vol. 15, 2003, pp. 145-181, doi:10.1207/S15327590IJHC1501_13.

[8] S. Chandrashekar, T. Stockman, D. Fels, and R. Benedyk, "Using think aloud protocol with blind users: a case for inclusive usability evaluation methods," In Proc. 8th Int. ACM SIGACCESS Conf. on Computers and accessibility (Assets '06). ACM, New York, NY, USA, 2006, pp. 251-252, doi:10.1145/1168987.1169040.

[9] H. Petrie, F. Hamilton, N. King, and P. Pavan, "Remote usability evaluations with disabled people," In Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI '06). ACM, New York, USA, 2006, pp. 1133-1141, doi:10.1145/1124772.1124942.

[10] A. Al-Wabil, and H. Al-Khalifa, "A framework for integrating usability evaluations methods: The Mawhiba web portal case study," In Current Trends in Information Technology (CTIT), 2009 Int. Conf. on the, IEEE, Dec. 2009, pp. 1-6, doi:10.1109/CTIT.2009.5423128.

[11] A. Blecken, D. Bruggemann, and W. Marx, "Usability evaluation of a Learning Management System," System Sciences (HICSS), 43rd Hawaii Int. Conf. on, Jan. 2010, pp. 1-9, doi:10.1109/HICSS.2010.422.

[12] "Just ask: integrating accessibility throughout design. Preparing for usability testing" - [online] Available at: http://www.uiaccess.com/accessucd/ut_prep.html. [Retrieved: Sept., 2014].

[13] M. Rowan, P. Gregor, D. Sloan, and P. Booth, "Evaluating web resources for disability access," In Proc. 4th ACM Int. Conf. on Assistive technologies (Assets '00), 2000, pp. 80-84, doi:10.1145/354324.354346.

[14] D. J. Mayhew, The usability lifecycle. A practitioner's handbook for user interface design, Morgan Kaufmann Publishers Inc., San Francisco, 1999.

[15] "UsabilityNet. User observation/field studies" – [online] Available at: http://www.usabilitynet.org/tools/userobservation.htm. [Retrieved: Sept., 2014].

[16] S. Riihiaho, "Experiences with usability evaluation methods," Licentiate thesis. Helsinki University of Technology. Laboratory of Information Processing Science, 2000.

[17] "Webbism - Web accessibility (Brisbane). The benefits of user testing with disabled users" - [online] Available at: http://webbism.com/2012/07/06/the-benefits-of-user-testing-with-disabled-users. [Retrieved: Sept., 2014].

[18] J. Karat, "User-centered software evaluation methodologies," In M. Helander, T.K. Landauer, P. Prabhu (Eds.) Handbook of human-computer interaction, 2nd ed. Elsevier Science. pp. 689-704, 1997.

[19] D. Rømen and D. Svanæs, "Evaluating web site accessibility: validating the WAI guidelines through usability testing with disabled users," In Proc. 5th Nordic Conf. on Human-computer interaction: building bridges (NordiCHI '08). ACM, New York, NY, USA, 2008, pp. 535-538, doi:10.1145/1463160.1463238.

[20] A. Lepistö and S. Ovaska, "Usability evaluation involving participants with cognitive disabilities," In Proc. 3rd Nordic Conf. on Human-computer interaction. ACM, NY, USA, 2004, pp. 305-308, doi:10.1145/1028014.1028061.

 ISBN: 978-1-61208-367-4

Towards Automating the Coherence Verification of Multi-Level Architecture Descriptions

Abderrahman Mokni*, Marianne Huchard†, Christelle Urtado*, Sylvain Vauttier* and Huaxi (Yulin) Zhang‡
*LGI2P, Ecole des Mines Alès, Nîmes - France
Email: {Abderrahman.Mokni, Christelle.Urtado, Sylvain.Vauttier}@mines-ales.fr
†LIRMM, CNRS and Université de Montpellier 2, Montpellier - France
Email: huchard@lirmm.fr
‡INRIA, Ecole Normale Supérieure de Lyon, Lyon - France
Email: yulinz88@gmail.com

***Abstract*—Component-Based Software Engineering considers off-the-shelf software component reuse as its cornerstone. In previous work, we proposed Dedal, a three level Architecture Description Language. It supports a novel modeling approach that aims at describing the specification, the implemented configuration and the running assembly of the software. This eases reuse by guiding the search for existing components. In this paper, we propose a formal approach that states the rules for component reuse and interoperability among Dedal models. The use of B, a specification language providing model-checking capabilities, enables the automatic verification of Dedal architecture descriptions. The approach is illustrated using the example of a home automation software.**

Keywords–Software architecture, component reuse, B formal models, component subtyping, component compatibility, architecture levels.

I. INTRODUCTION

Component-Based Software Engineering (CBSE) aims at engineering software from previously developed components. Expected outcomes are to increase development speed and software quality, to ease the development of software of ever increasing complexity and to decrease costs. In previous work [1], we proposed a three step approach to specify, design and deploy software architectures from existing software components. This proposal also includes means to control architecture evolution. It is supported by a three level Architecture Description Language (ADL) and component model called Dedal. The originality of this approach is to focus on component reuse by guiding the search for adequate components during the CBSE process. In this paper, we propose rules to automatically support verification and validation of Dedal's architecture descriptions which is a first step to handle reuse and architecture-centric evolution in a rigorous way. The rules are expressed in the B [2] notation, a formalism that can be automatically verified using existing provers and model checking tools. The remaining of the paper is structured as follows. Section II gives an overview of the three Dedal models and illustrates them with a home automation architecture example. Section III presents an overview of our formalization of Dedal models using the B notation. Section IV sets the intra-level rules for component substitutability and compatibility. Section V describes inter-level rules that define the relations between component descriptions in two successive description levels. Section VI depicts an overview of the experimentation of the presented formal models and rules. Section VII analyzes related work before Section VIII concludes and discusses future work.

II. OVERVIEW OF THE DEDAL MODEL

Dedal is an ADL that helps software development at three abstraction levels. These levels have been designed to support reuse-centered architecture development. In the following, we detail each of Dedal's three abstraction levels [1]. To illustrate these concepts, we propose to model a part of Home Automation Software (HAS) that manages comfort scenarios. Here, it automatically controls the building's lighting and heating in function of the time and ambient temperature. For this purpose, we propose an architecture with an orchestrator component that interacts with the appropriate devices to implement the scenario.

A. The abstract architecture specification

The *abstract architecture specification* is the first level of architecture software descriptions. It provides a generic view of the global structure of the software and states its expected functionalities according to functional requirements. An architecture specification may correspond to a prescriptive architecture, which describes the system's architecture "as-wished" at specification time, as defined by Taylor *et al.* [3]. In Dedal, the architecture specification is composed of component roles and their connections. *Component roles* represent the roles that components are expected to play in the system. They thus are abstract and partial component type specifications. They are identified by the architect in order to search for and select corresponding concrete components in the next step. Figure 1-a shows a possible architecture specification for the HAS. In this specification, five component roles are identified. A component playing the *HomeOrchestrator* role controls four components playing the *Light*, *Time*, *Thermometer* and *CoolerHeater* roles.

B. The concrete architecture configuration

The *concrete architecture configuration* is an implementation view of the software architecture. It results from the selection of existing component classes in component repositories. Thus, an architecture configuration lists the concrete

 ISBN: 978-1-61208-367-4

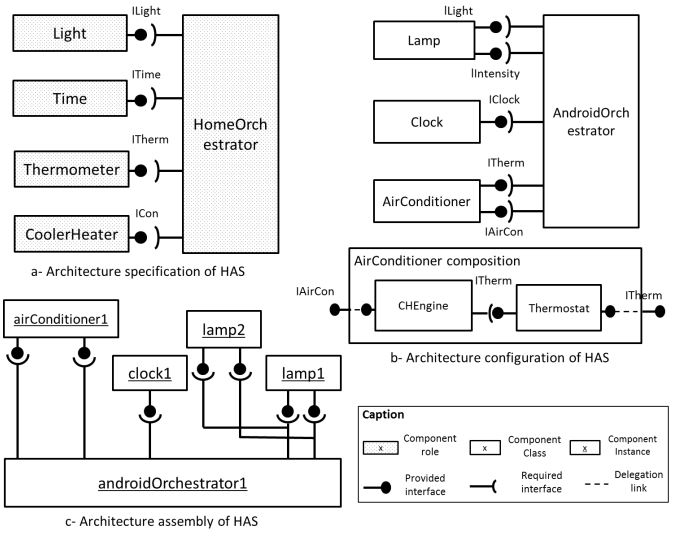


Figure 1: Architecture specification, configuration and assembly of HAS

component classes that compose a specific version of the software system.

Component classes are concrete component implementations. In Dedal, component classes can be either primitive or composite. *Primitive component classes* encapsulate executable code. *Composite component classes* encapsulate an inner architecture configuration (*i.e.,* a set of connected component classes which may, in turn, be primitive or composite). A composite component class exposes a set of interfaces corresponding to unconnected interfaces of its inner components. A *Component type* gives an abstract representation of a set of component classes. It defines the set of interfaces that a class must hold to be an implementation of this type. Component types are used to classify component classes and build indexes on the content of component repositories. To search for component classes that can be used to implement an architecture specification, component roles are matched with component types (using a classification based on specialization and substitutability in a manner similar to Arévalo *et al.* [4]). As they are implementations of their declared component types, these component classes are valid realizations of the component roles. Figure 1-b shows the architecture configuration of the HAS example as well as an example of an *AirConditioner* composite component and its inner configuration. As illustrated in this example, a single component class may realize several roles in the architecture specification as with the *AirConditioner* component class which realizes both *Thermometer* and *CoolerHeater* roles. Moreover, a component class may provide more services than those listed in the architecture specification as with the *Lamp* component class that provides an extra service to control the intensity of light.

C. The instantiated architecture assembly

The *instantiated architecture assembly* describes software at runtime and gathers information about its internal state. The architecture assembly results from the instantiation of an architecture configuration. It lists the instances of the component and connector classes that compose the deployed architecture at runtime and their assembly constraints (such as maximum numbers of allowed instances).
Component instances document how component classes from an architecture configuration are instantiated in the software. Each component instance has an initial and current state represented by a list of valued attributes. Figure 1-c shows an instantiated architecture assembly for the HAS example.

D. Motivation

The three-level Dedal model is a novel approach to component-based software development that increases reuse by supporting the search for off-the-shelf components. The associated ADL focuses on the description of architectural concepts in three separated abstraction levels but it lacks mechanisms to verify and validate architecture definitions before and after evolution. This work aims to provide mechanisms to automate the verification and validation of coherence between architecture levels from requirement to runtime. We propose a set of rules to define the relations inside each abstraction level and between two of them. The rules are expressed using B [2], a first-order logic and set-theoretic language with a rich expressiveness that can be automatically verified using existing model checkers.

III. OVERVIEW OF THE FORMALIZATION

The formalization is divided into two parts. A first part, which is generic and independent from any architectural model, consists in formalizing the most common concepts of software architectures. The second part is specific to Dedal and consists in formalizing concepts and relationships of the Dedal model. The formal model of Dedal therefore is a specialization of the generic model. In the remainder, we present the most important parts of the formalization.

A. Formalizing underlying architectural constructs

During the last decades, a consensus established that architectures were made of three main elements [5]: components (loci of computation), connectors (mediators) and configurations (topologies of the architecture). In Table I, we give the formal definition and relations between these concepts (the *arch_concepts* model).

We note that *Arch_concepts* includes an inner model called *Basic_concepts* which contains the formalization of finer grained elements (*i.e.,* interfaces and signatures). *Basic_concepts* is not presented in this paper for the sake of space.

B. Formalization of Dedal architecture levels

Dedal proposes three abstraction levels to describe architectures. Formalizing each of these levels enables to verify each of them separately but also to check the global coherence of architecture definitions.

The *Arch_specification* model. An architecture specification inherits from the generic definition of an architecture as stated in the *Arch_concepts* model. In Dedal, an architecture specification is specifically made of a set of component roles. Roles are thus defined as specializations of components by the following property: $COMP\_ROLES \subseteq COMPS \wedge compRole \subseteq COMP\_ROLES$.

The *Arch_configuration* model. In the same way, the component class concept used in the *Arch_configuration* model

 ISBN: 978-1-61208-367-4

TABLE I: Formal specification of underlying concepts

MACHINE $Arch\_concepts$
INCLUDES $Basic\_concepts$
SETS
$ARCHS; COMPS; COMP\_NAMES$
VARIABLES
$architecture, arch\_components, arch\_connections, component, comp\_name, connection, comp\_interfaces, client, server$
INVARIANT
/* A component has a name and a set of interfaces */
component $\subseteq$ *COMPS* $\wedge$
comp_name $\in$ *component* $\rightarrow$ *COMP_NAMES* $\wedge$
comp_interfaces $\in$ *component* $\rightarrowtail$ $\mathcal{P}$ *(interface)* $\wedge$
/* A client (resp. server) is a couple of a component and an interface */
client $\in$ *component* $\leftrightarrow$ *interface* $\wedge$
server $\in$ *component* $\leftrightarrow$ *interface* $\wedge$
/* A connection is a one-to-one mapping between a client and a server */
connection $\in$ *client* $\rightarrowtail\!\!\!\!\!\!\rightarrow$ *server* $\wedge$
/* An architecture has a set of components and connections */
architecture $\subseteq$ *ARCHS* $\wedge$
arch_components $\in$ *architecture* $\rightarrow$ $\mathcal{P}$ *(component)* $\wedge$
arch_connections $\in$ *architecture* $\rightarrow$ $\mathcal{P}$ *(connection)*

Specific B notations:
$\rightarrow$: total function $\leftrightarrow$: relation $\rightarrowtail$: injection
$\rightarrowtail\!\!\!\!\!\!\rightarrow$: bijection $\mathcal{P}$(<set>): powerset of <set>

is defined as a specialization of the component concept, as they share the same properties (name, interface, etc.). Table II shows the formalization of the configuration level.

TABLE II: Formal specification of the configuration level

MACHINE $Arch\_configuration$
INCLUDES $Arch\_concepts, Arch\_specification$
SETS
COMP_CLASS; *CLASS_NAME*; *ATTRIBUTES*; *CONFIGURATIONS*
CONSTANTS
COMP_TYPES
PROPERTIES
/* Component types are also a specialization of components distinct from roles */
COMP_TYPES $\subseteq$ *COMPS* $\wedge$ *COMP_TYPES* = *COMPS* - *COMP_ROLES*
VARIABLES
config, *config_components*, *config_connections*, *compType*, *compClass*, *class_name*, *class_attributes*, *compositeComp*, *delegatedInterface*, ...
INVARIANT
compType $\subseteq$ *COMP_TYPES* $\wedge$
/* A component class has a name and a set of attributes */
compClass $\subseteq$ *COMP_CLASS* $\wedge$ *class_name* $\in$ *compClass* $\rightarrow$ *CLASS_NAME* $\wedge$
attribute $\subseteq$ *ATTRIBUTES* $\wedge$ *class_attributes* $\in$ *compClass* $\rightarrow$ $\mathcal{P}$*(attribute)* $\wedge$
/* A composite component has also a configuration and is constituted of component classes */
compositeComp $\subseteq$ *compClass* $\wedge$ *composite_uses* $\in$ *compositeComp* $\rightarrow$ *config* $\wedge$
/* A delegation is a mapping between a delegated interface and its corresponding one */
delegatedInterface $\subset$ *interface* $\wedge$
delegation $\in$ *delegatedInterface* $\rightarrowtail$ *interface* $\wedge$
/* A configuration is a set of component classes and connections*/
config $\subseteq$ *CONFIGURATIONS* $\wedge$
config_components $\in$ *config* $\rightarrow$ $\mathcal{P}$*(compClass)* $\wedge$
config_connections $\in$ *config* $\rightarrow$ $\mathcal{P}$*(connection)*

The *Arch_assembly* model. The *Arch_assembly* model captures the definition of architectures at the instance level. Component instances are mapped to initial and current states. This information is useful to audit software evolution at runtime and control dynamic reconfigurations. Next section sets Dedal's intra-level rules by defining invariant constraints on the previously defined concepts.

IV. INTRA-LEVEL RULES

A. Component substitutability rules

In software architectures, substitutability determines when a component can replace another while holding the architecture consistent. The notion of substitutability was firstly discussed in object-oriented languages to define subtyping and object interoperability. This notion has also been discussed in the component-based paradigm [6] [7] but there is still no consensus in defining component substitutability. Indeed, components are complex entities that can be studied from many views (syntactic, semantic, protocol, etc.). In Dedal, at least a syntactic substitutability is needed to filter components while searching for suitable ones in repositories. The corresponding rules can be extended later to take dynamic behavior into account. Figure 2 shows an example of component subtyping that illustrates the main substitutability rules. The principle that is enforced is that a subtype should provide at least the same services as its supertype and require the same or less services. For example, *Clock* can be substituted for *ClockV2* which, provides one more interface (*IInfo*) and requires one less interface (interface *ILanguage* is no more required) (*cf.* Rule 1), and the interface type *ILocation* is subtyped by *ILocation&GMT* which has one more signature *getGMT()* (*cf.* Rule 3).

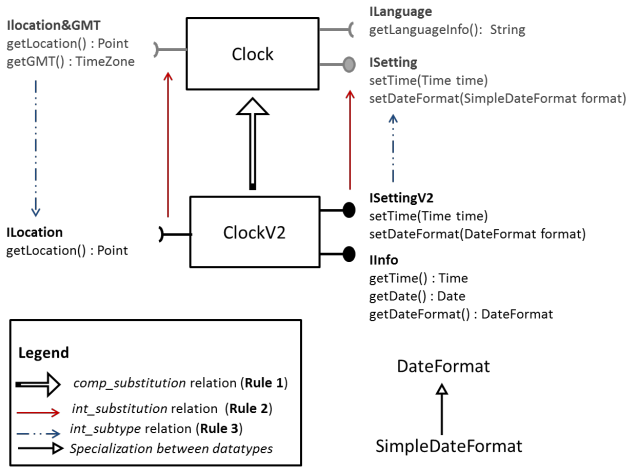


Figure 2: Example of component substitutability

Rule 1: Component substitutability. A component *C_sup* can be substituted for a component *C_sub* iff there exists an injection *inj1* between the set of interfaces of *C_sup* and the set of interfaces of *C_sub* such that *int* can be substituted for *inj1*(*int*), *int* being a provided interface of *C_sup*, and there exists an injection *inj2* between the set of interfaces of *C_sub* and the set of interfaces of *C_sup* such as *inj2*(*int*) can be substituted for *int*, *int* being a required interface of *C_sub*. Formally:

$comp\_substitution \in component \leftrightarrow component \wedge$
$\forall$ *(C_sup , C_sub).*
(C_sup $\in$ *component* $\wedge$ *C_sub*$\in$ *component* $\wedge$ *C_sup* $\neq$ *C_sub*
$\Rightarrow$
(C_sub $\in$ *comp_substitution [{ C_sup}]*
$\Leftrightarrow$
$\exists$ *(inj1 , inj2).*
(inj1 $\in$ *providedInterfaces (C_sup)* $\rightarrowtail$ *providedInterfaces(C_sub)* $\wedge$
$\forall$ *(int).*
(int $\in$ *interface* $\wedge$ *int* $\in$ *providedInterfaces(C_sup)*
$\Rightarrow$
inj1(int) $\in$ *int_substitution [{int}])* $\wedge$

inj2 $\in$ *requiredInterfaces (C_sub)* $\rightarrowtail$ *requiredInterfaces (C_sup)* $\wedge$
$\forall$ *(int).*
(int $\in$ *interface* $\wedge$ *int* $\in$ *requiredInterfaces (C_sub)* $\wedge$
$\Rightarrow$
int $\in$ *int_substitution [{inj2 (int)}])))*

According to Rule 1, the component subtype can have more provided and fewer required interfaces than its supertype. This rule depends on interface substitutability which we define as follows:

Rule 2: Interface substitutability. Interface substitutability depends on the interface type and direction. Interface substyping is given by Rule 3. When both interfaces are provided, substitutability is covariant with interface subtyping (*i.e.,* a provided interface *int_sup* is substituted for a provided interface *int_sub* iff the type of *int_sub* is a subtype of *int_sup*'s type). In the second case where the two interfaces are required, substitutability is contravariant with interface subtyping (*i.e.,* a required interface *int_sup* is substituted for a required interface *int_sub* iff the type of *int_sup* is a subtype of *int_sub*'s type).

Rule 3: Interface subtyping. An interface type *intTypeSub* is a subtype of an interface type *intTypeSup* iff there exists an injection *inj* between the signature set of *intTypeSup* and the signature set of *intTypeSub* such that for each signature *sig* of *intTypeSup*, *inj*(*sig*) specializes *sig*.

$$
\begin{array}{l}
int\_subtype \in interfaceType \leftrightarrow interfaceType \wedge \\
\forall\ (intTypeSup, intTypeSub). \\
(intTypeSup \in interfaceType \wedge intTypeSub \in interfaceType \wedge \\
intTypeSup \neq intTypeSub \\
\quad \Rightarrow \\
\quad ((intTypeSup, intTypeSub) \in int\_subtype \\
\quad\quad \Leftrightarrow \\
\quad\quad \exists\ inj. \\
\quad\quad (inj \in int\_signatures(intTypeSup) \rightarrowtail int\_signatures(intTypeSub) \wedge \\
\quad\quad\quad \forall\ (sig). \\
\quad\quad\quad (sig \in signature \wedge sig \in int\_signatures(intTypeSup) \\
\quad\quad\quad\quad \Rightarrow \\
\quad\quad\quad inj(sig) \in sig\_subtype[\{sig\}])) \\
\quad) \\
)
\end{array}
$$

According to Rule 3, interface subtyping allows to add new signatures. This is why this relation is used both in a covariant way on provided interfaces and in a contravariant way (to require less signatures) on required interfaces in Rule 2. Interface subtyping in turn relies on signature specialization.

Rule 4: Signature specialization. Signature specialization conforms to method overriding in the object-oriented paradigm. A specialized signature must have contravariant specialization of parameter types and covariant specialization of return type as it must require less information when invoked and provide richer results. To define signature specialization, we first consider parameter specialization.

Rule 4.1. A signature *sig_sub* is parameter subtype of a signature *sig_sup* iff there exists an injection *inj* between the parameters of *sig_sup* and the parameters of *sig_sub* and for each parameter *param* of *sig_sup*, *inj*(*param*) has the same name as *param* and the type of *inj*(*param*) is a subtype of *param*'s type.

$$
\begin{array}{l}
\forall\ (sig\_sup, sig\_sub). \\
(sig\_sup \in signature \wedge sig\_sub \in signature \wedge sig\_sup \neq sig\_sub \\
\Rightarrow (\\
\quad (sig\_sup, sig\_sub) \in param\_subtype \\
\quad \Leftrightarrow \\
\quad \exists\ inj.(inj \in parameters(sig\_sup) \rightarrowtail parameters(sig\_sub) \wedge \\
\quad\quad \forall\ param.(param \in parameter \wedge \\
\quad\quad\quad param \in parameters\ (sig\_sup) \\
\quad\quad \Rightarrow \\
\quad\quad param\_name\ (param) = param\_name\ (inj\ (param)) \wedge \\
\quad\quad parameter\_type\ (inj(\ param\ )) \\
\quad\quad\quad \in closure\ (subtype)[\{parameter\_type\ (param)\}])) \\
\quad\quad) \\
\quad)
\end{array}
$$

Rule 4.2. A signature *sig_sub* specializes a signature *sig_sup* if and only if they have the same name and *sig_sup* is parameter subtype of *sig_sub* and the return type of *sig_sub* is a subtype of the return type of *sig_sup*.

$$
\begin{array}{l}
\forall\ (sig\_sup, sig\_sub). \\
(\ sig\_sup \in signature \wedge sig\_sub \in signature \wedge sig\_sup \neq sig\_sub \\
\Rightarrow (\\
\quad (sig\_sup, sig\_sub) \in sig\_subtype \\
\quad \Leftrightarrow (\\
\quad\quad sig\_name\ (sig\_sup) = sig\_name\ (sig\_sub) \wedge \\
\quad\quad (sig\_sub, sig\_sup) \in param\_subtype \wedge \\
\quad\quad sig\_return\ (sig\_sub) \in closure\ (subtype)[\{sig\_return\ (sig\_sup)\}]) \\
\quad)
\end{array}
$$

B. Component compatibility rules

Components compatibility relies on interface compatibility. Two components can interact if and only if they have at least two compatible (connectable) interfaces.

Rule 5: Interface compatibility. A provided interface *intA* and a required interface *intB* are compatible iff the type of *intA* is a subtype of *intB*'s.

In other words, a provided interface should declare the same, a specialization of and possibly extra signatures than the required interface to ensure that all the required functionalities can be supplied.

Substitutability and compatibility rules are defined for general-purpose. In Dedal, they are used to check intra-level relations between components of the same kind (*i.e.,* roles, types, classes or instances). In the remainder, we focus on the inter-level rules which enable to check the global coherence between the multiple architecture definitions.

V. INTER-LEVEL RULES

Specifying inter-level rules is a crucial step to ensure coherence between architecture levels from requirements to runtime (*cf.* Figure 3). In order to go from the specification of an architecture to an implemented configuration, the architect must select suitable concrete component classes that realize the specified roles. The implementation can then be instantiated and deployed in multiple contexts. Inter-level rules are the foundations to automate such a reuse process in component-based software development.

A. Relations between the specification and configuration levels

Two main relations are considered between the specification and configuration levels: the matching relation

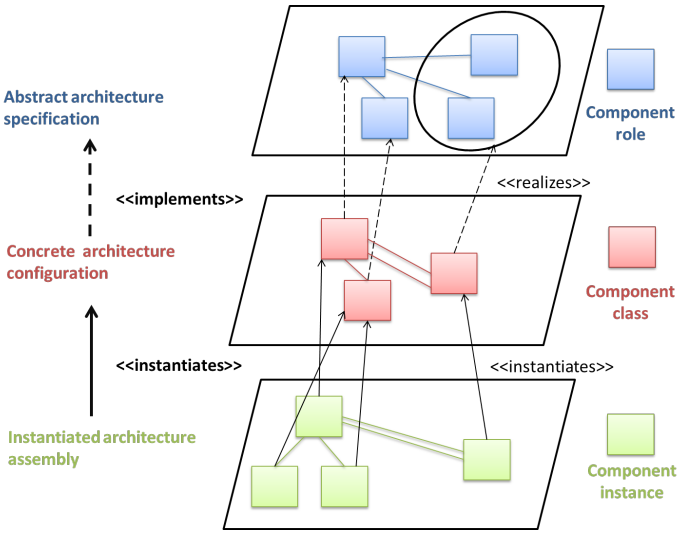


Figure 3: Relations between architecture levels

between component roles and concrete component types and the realization relation between component roles and component classes.

Rule 6: Component type matching. A component type *CT* matches with a component role *CR* iff it exists an injection *inj* between the set of interfaces of *CR* and the set of interfaces of *CT* such that *int* can be substituted for *inj*(*int*), *int* being an interface of *CR*. Formally:

$$
\begin{aligned}
&matches \in compType \leftrightarrow compRole \wedge \\
&\forall\ (CT,\ CR).(CT \in compType \wedge CR \in compRole \\
&\Rightarrow \\
&((CT,CR) \in matches \\
&\quad \Leftrightarrow \\
&\exists (inj).(inj \in comp\_interfaces\ (CR) \rightarrowtail comp\_interfaces\ (CT) \wedge \\
&\qquad \forall\ (int).(int \in interface \Rightarrow inj\ (int) \in int\_substitution\ [\{int\}]) \\
&)))
\end{aligned}
$$

As stated in Section II, component role descriptions are specified by the architect to guide the search for existing component classes. Hence, there are several ways to find a concrete realization of component roles. A component class can realize several roles at once or a composition of component classes (composite component) can complement each other to realize a given role. This holds a many-to-many mapping between component roles and concrete component types.

Rule 7: Component implementation. To draw an analogy with object-oriented programming, the relation between a component class and a component type is similar to the relation between a class and an interface. A component class must implement all the provided interfaces of the component type. Implementation details (that depend on decisions of the architect) are out of the scope of the abstract aspects that we intend to validate. However, an abstract formalization of the implementation is compulsory to make our formal model coherent. Component implementation is defined as follows:

$$
class\_implements \in compClass \rightarrow compType
$$

Rule 8: Component realization. The relation between a component class and a component role is a corollary of the matching relation (Rule 6) and the implementation relation (Rule 7). Indeed, a component class *CL* realizes a component role *CR* iff it exists a component type *CT* implemented by *CL* and that matches with *CR*. Formally:

$$
\begin{aligned}
&realizes \in compClass \leftrightarrow compRole \wedge \\
&\forall\ (CL,\ CR).(CL \in compClass \wedge CR \in compRole \\
&\Rightarrow \\
&\quad ((CL,\ CR) \in realizes \\
&\qquad \Leftrightarrow \\
&\quad \exists\ CT.(CT \in compType \wedge (CT\ ,\ CR) \in matches \wedge \\
&\quad (CL,CT) \in class\_implements)) \\
&)
\end{aligned}
$$

Rule 9: Relation between an architecture specification and its configuration. An architecture configuration *Conf* realizes an architecture specification *Spec* iff for each component role *CR* in *Spec* it exists a component class *CL* in *Conf* such that *CL* realizes *CR*.

$$
\begin{aligned}
&implements \in config \leftrightarrow arch\_spec \wedge \\
&\forall\ (Conf,\ Spec).(Conf \in config \wedge Spec \in arch\_spec \\
&\quad \Rightarrow \qquad (Conf,\ Spec) \in implements \\
&\qquad \Leftrightarrow \\
&\quad \forall\ CR.(CR \in compRole \wedge CR \in spec\_components\ (Spec) \Rightarrow \\
&\qquad \exists\ CL.(CL \in compClass \wedge CL \in config\_components\ (Conf) \wedge \\
&\qquad (CL,\ CR) \in realizes)))
\end{aligned}
$$

B. Relation between the configuration and assembly levels

An architecture assembly is composed of instances of the component classes that are in the architecture configuration. The instantiation depends on many technical choices that should be made by the architect (*e.g.,* the choice of the runtime framework) and should not be considered at such an abstract level of formalization.

$$
comp\_instantiates \in compInstance \rightarrow compClass
$$

The instantiation is a total function between the set of component instances and the set of component classes. This means that each component instance instantiates one and only one component class. Conversely, a component class can have several instances (the number of instances can be specified through assembly constraints).

Consequently, an architecture assembly *Asm* instantiates an architecture configuration *Conf* iff every component class *CL* of *Conf* is instantiated at least once by a component instance *CI* in *Asm* and every component instance *CI* in *Asm* is an instance of a component class in *Conf*:

$$
\begin{aligned}
&instantiates \in assm \rightarrow config \\
&\forall\ (Asm,Conf).(Asm \in assm \wedge Conf \in config \\
&\Rightarrow \\
&((Asm,Conf) \in instantiates \\
&\quad \Leftrightarrow \\
&\forall\ CL.(CL \in compClass \wedge CL \in config\_components(Conf) \\
&\quad \Rightarrow \\
&\quad \exists\ CI.(CI \in compInstance \wedge CI \in assm\_components(Asm) \wedge \\
&\quad (CI,CL) \in comp\_instantiates) \wedge \\
&\forall\ CI.(CI \in compInstance \wedge CI \in assm\_components(Asm) \\
&\quad \Rightarrow \\
&\quad \exists\ CL.(CL \in compClass \wedge CL \in config\_components(Conf) \wedge \\
&(CI,CL) \in comp\_instantiates)))))
\end{aligned}
$$

VI. Experimentation overview

In order to validate the proposed rules, formal models are manually instantiated using simple tests covering the main cases. Each test corresponds to a specific instantiation of

 ISBN: 978-1-61208-367-4

a given architectural model to check if one of the defined rules meets the required definition. Models are checked using ProB [8], a model checker of B that shows invariant violations and the current state of the given model. In case a violation is detected, either instantiation is wrong or the defined rules have to be revisited. At this stage of work, all rules have been manually validated and can be used later to automate the analysis process.

In future work, we aim to automatically generate the specification of model instances from the graphical or textual descriptions of architectures. The derived models will then be passed to the model checker for automatic verification of the architectural descriptions.

VII. Related work

Over two decades ago, many researches focused on giving ADL's a formal representation. A classification of the major ADL's was proposed by Medvidovic *et al.* [5]. Although, most of these ADL's provide the required features to describe an architecture, they often are either domain-specific or lack formal foundations to support automatic analysis and dynamic evolution. Some ADL's like Wright [9] and Rapide [10] focus on the specification and verification of architectural behavior. Wright uses CSP, a formal language based on process algebra while Rapide uses partially ordered sets (posets) of events to model behavior and enable formal reasoning on architecture specifications. Both Wright and Rapide, however, focus on architecture behavior and do not consider its structure. They do not provide any mechanism for component reuse and do not support multiple abstraction levels either.

Other close works are the formalization and analysis of architectural styles using a formal language. Kim and Garlan [11] propose an approach for modeling and analyzing architectural styles using Alloy. These works address architecture styles rather than architecture constructs and aim to provide a generic formal model for several styles like C2 [12] or the pipe and filter style. Our focus is slightly different since we address the structure of architectures independently from its style.

Our work has also drawn inspiration from type theory in object languages [13]. Like objects, components can have subtyping relations that enable reuse and software evolution. However, components are more complex than plain objects and they do not obey the same rules. To our knowledge, there was no real attempt, except for Medvidovic *et al.* to adapt the theory of objects to components. Medvidovic *et al.* propose a type theory for software architectures by multiple component subtyping and have the architect decide about which properties (name, interface, behavior or implementation) he wants to specialize. They applied their theory on their C2SADEL [6] ADL. In our three level component model, we need different typing rules to define relations between components into and between each levels of architecture descriptions. A part of our subtyping rules is also inspired from our previous work on building component directories using Formal Concept Analysis [14]. In fact, rules for specializing functionality signatures were defined to guide the search for compatible or substitutable components in a yellow-page like component directory.

VIII. Conclusion and future work

This paper proposes mechanisms to automate component reuse and inter-level coherence checking in a component-based development process. The outlined approach consists in coupling a three-level ADL called Dedal with B formal models. These models were reinforced with invariant constraints to set substitutability and compatibility rules into each abstraction level and inter-level rules to enable (1) reuse by guiding the search for concrete component classes and (2) coherence checking between abstraction levels. This work sets the basis for the definition of evolution rules which will be the next step of our contribution. Indeed, the proposed mechanisms will be used to automatically handle software evolution and propagate changes among architecture descriptions to preserve coherence.

A practical issue of future work will be to provide a toolset for Dedal, our three-level ADL. Indeed, we plan to map Dedal to UML and provide a visual modeling tool. Furthermore, we are considering the integration of existing model checkers and animation tools to automate verification and realize simulations and early validations of evolution scenarios.

References

[1] H. Y. Zhang, C. Urtado, and S. Vauttier, "Architecture-centric component-based development needs a three-level ADL," in Proc. of the 4th ECSA, ser. LNCS, vol. 6285. Copenhagen, Denmark: Springer, August 2010, pp. 295–310.

[2] J.-R. Abrial, The B-book: Assigning Programs to Meanings. Cambridge University Press, 1996.

[3] R. Taylor, N. Medvidovic, and E. Dashofy, Software architecture: Foundations, Theory, and Practice. Wiley, 2009.

[4] G. Arévalo, N. Desnos, M. Huchard, C. Urtado, and S. Vauttier, "FCA-based service classification to dynamically build efficient software component directories," International Journal of General Systems, 2008, pp. 427–453.

[5] N. Medvidovic and R. N. Taylor, "A classification and comparison framework for software architecture description languages," IEEE TSE, vol. 26, no. 1, Jan. 2000, pp. 70–93.

[6] N. Medvidovic, D. S. Rosenblum, and R. N. Taylor, "A language and environment for architecture-based software development and evolution," in Proc. of the 21st ICSE, Los Angeles, USA, 1999, pp. 44–53.

[7] A. Beugnard, J.-M. Jezequel, N. Plouzeau, and D. Watkins, "Making components contract aware," Computer, vol. 32, no. 7, Jul 1999, pp. 38–45.

[8] M. Leuschel and M. Butler, "ProB: An automated analysis toolset for the B method," International Journal on Software Tools for Technology Transfer, vol. 10, no. 2, Feb. 2008, pp. 185–203.

[9] R. Allen and D. Garlan, "A formal basis for architectural connection," ACM TOSEM, vol. 6, no. 3, Jul. 1997, pp. 213–249.

[10] D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Vera, D. Bryan, and W. Mann, "Specification and analysis of system architecture using rapide," IEEE TSE, vol. 21, 1995, pp. 336–355.

[11] J. S. Kim and D. Garlan, "Analyzing architectural styles," Journal of Systems and Software, vol. 83, no. 7, Jul. 2010, pp. 1216–1235.

[12] R. N. Taylor, N. Medvidovic, K. M. Anderson, E. J. Whitehead, Jr., and J. E. Robbins, "A component- and message-based architectural style for GUI software," in Proc. of the 17th ICSE. Seattle, USA: ACM, 1995, pp. 295–304.

[13] B. H. Liskov and J. M. Wing, "A behavioral notion of subtyping," ACM TOPLAS, vol. 16, no. 6, 1994, pp. 1811–1841.

[14] N. A. Aboud, G. Arévalo, J.-R. Falleri, M. Huchard, C. Tibermacine, C. Urtado, and S. Vauttier, "Automated architectural component classification using concept lattices," in Proc. WICSA/ECSA, Cambridge, UK, September 2009, pp. 21–31.

 ISBN: 978-1-61208-367-4

A Set-Oriented Formalism as a Foundation for the Modeling and Verification of Connected Data and Process Specifications

Julia Martini, Hannes Restel, Raik Kuhlisch, Jörg Caumanns
E-HEALTH // ESPRI
Fraunhofer-FOKUS
Berlin, Germany
{julia.magdalena.martini, hannes.restel, raik.kuhlisch, joerg.caumanns}@fokus.fraunhofer.de

Abstract—**In this work-in-progress report, a methodology based on a fully formalized and machine-readable formalism is introduced. The goal is to help modelers/developers to design and verify specifications, standards, and profiles in the field of information exchange. The formalism allows the specification and verification of process models as well as data/information models. These formalized specifications describe the structure (syntax) and meaning (semantics) of data models and process models as well as relations (requirements, dependencies, rules, constraints, pre-/post-conditions) between them. Unlike the traditional approach of defining a specification, which is to first write an unstructured specification document and then to derive a platform-specific binding from it (e.g., XML Schema), the specification itself is directly defined in a structured and machine-understandable formalism on a logical level. Fully formalized specifications allow for automatic validation and verification and, therefore, allow checking if the specification is complete and consistent so that dependencies between process steps can be verified. This work in progress lines out the very foundations of the described methodology by introducing a Set-Oriented formalism (SOF) that is used to formalize data models and dependencies.**

Keywords-specification; formalism; profiling; validation; information modeling.

I. INTRODUCTION

To simplify the development of applications and to achieve interoperability, acknowledged specifications (de jure standards, de facto standards) for document and message exchange are widely used in our current era of net-based information exchange. An information exchange specification may define data and information models as well as process/protocol models. To support a multitude of different use cases in a variety of domains, standards are usually defined in a rather generic way. This often results in an (intentionally) ambiguous specification that allows multiple interpretations by different parties and, therefore, limits interoperability. To counter this effect, domain specific profiles are derived to restrict the specification and to make it unambiguous. Even a set of specifications may be compiled into a single profile to specify a more complex process.

Often, the existing information exchange specifications are barely represented as fully formalized documents and, therefore, cannot be understood by machines. They need to be manually interpreted, transformed and bound into a serializable, machine-computable representation on the platform specific level [1] that is finally used to generate and exchange instances (messages, documents) of those models on runtime. Technologies/methodologies are widely used to support those steps (see Table I).

For example, the Unified Modeling Language (UML) [2] and the Object Constraint Language (OCL) [3] may be used to define the data models as well as constraints, and the Extensible Markup Language (XML) Schema [4] and Schematron [5] may be used for the binding. Still, large parts of the specifications located on the computational independent level (CIM, see [1]) and platform independent level (PIM, see [1]) are represented as unstructured (free text) documents describing the purpose, syntax and semantics of a specification. Thus, an automated transformation from one step of the specification development chain to the other is rarely possible.

If a set of specifications is compiled to define a profile, then the complexity increases because relations (requirements, dependencies, rules, constraints, pre-/ post-conditions) between the data and process models of each incorporated specification do exist. The more complex those relations are, the more difficult it is to define a valid, complete and unambiguous profile and to verify the correctness of the profile.

To counteract the above-mentioned problems, a methodology based on a mathematical, formalized and machine-readable formalism is introduced in this report, called Set-Oriented formalism (SOF).

This formalism allows the specification and verification, both of the defined process models as well as the data/information models of a specification including the relations between the models on a level prior to the platform specific level, i.e., on CIM and PIM.

 ISBN: 978-1-61208-367-4

TABLE I. DEVELOPMENT CHAIN FROM SPECIFICATION OVER BINDING TO INSTANCES

Step	Denoted in
Specification (Standard, Profile)	Unstructured document, UML/OCL, Business Process Modeling Language, Fundamental Modeling Concepts (FMC) etc.
Binding	XML Schema, Schematron, Resource Description Framework-Schema (RDFS), Structured Query Language (SQL), JSR-94, etc.
Instance	XML, JavaScript Object Notation (JSON), etc.

In Section II, existing formalisms are evaluated, especially UML and OCL. In Section III, the Set-Oriented Formalism itself is introduced by formally defining its structure component and rule component. Subsequently, the transformation from a sample model depicted in SOF into a platform specific perspective (i.e., XML/Schematron representation) is described in Section IV. To support comprehensibility, all Sections make use of a shared example. This report concludes with a short summary of the findings and an outlook (Section V).

II. RELATED WORK

The worldwide established modeling language UML supports the standardized specification, construction and documentation of a system. UML's boundaries are that element-spanning semantic constraints/dependencies are not representable. For defining these rules for model elements, UML was augmented with the OCL, which is a declarative language. With the OCL one can, for example, define invariants, pre-, and post-conditions.

In SOF, the elementary dependencies regarding the cardinalities and data types that the UML depicts graphically are expressed with the structure component. The rules for model elements (e.g., invariants, pre- and post-conditions) are defined in the rule component, thereby interpreting each rule as a set of sets, which constitutes a valid instance of a model regarding that rule. OCL's boundaries are that inconsistent specifications, that is the combination of constraints contradicting each other, cannot be recognized. In SOF, the recognition of inconsistent specifications is possible. Since each constraint represents a set of valid instances of the model, checking via the intersecting set can determine whether an instance exists at all that fulfills all constraints (see Section III.C, list item c). In particular, this can be done pairwise for the constraints. The criteria for such a consistent specification is, hence, that the intersecting set of constraint sets is not empty. The familiar "frame problem" that can arise with OCL can also be solved with SOF since the post-conditions can be represented in SOF as the final state of the whole system.

III. THE SET-ORIENTED FORMALISM (SOF)

The SOF is designed to specify fully formalized and serializable data/ information models. Any data model, which can be transformed into a tree, is defined as a serializable information model. The basic principle of SOF is to represent all elements and their properties and constraints of a serializable information model as a set. For example, dependencies regarding the cardinality, data type or value of each element are represented as a set. This enables the interpretation of element-spanning constraints as a set of sets that constitutes a valid instance of the model regarding the constraint. Through this set-oriented representation of each element and the constraints, each verification problem can be reduced to a subset or intersecting-set problem (see Section III.C). The SOF has two main components: the structure and the rule component. The structure component is a degenerated table. Each cell represents an element of the information model referenced by an identifier. Using these identifiers, rules defined within the rule component may be attached to each cell to express dependencies regarding the values and cardinalities of the elements among themselves.

A. The Structure Component

The structure component itself includes a constructively defined table T_S that represents a serializable information model S in a way that each cell unit (called T_S-cell) is a representative of a unique element of S. Each T_S-cell is fully formalized, providing information about the data types and cardinalities of its S-element representative. The construction is hierarchy-preserving, so that each T_S-cell is assigned to a unique identifier i by its location. The identifier i is assigned to the same element of the information model using an algorithm to navigate within the tree of the information model (as shown in Figure 5 below).

1) Construction of T_S

Assume a serializable information model S and an infinite table T so that T contains an infinite amount of rows and columns. Additionally, the tuple (i, j) with $i, j \in \mathbb{N}^+$ will be the representative of the T-cell that is located in the i-th row and j-th column. The algorithm for the construction of T_S is displayed in Figure 1, whereby the infinite table T evolves into T_S, as the representative of the information model S.

Each time *createStructuralComponent()* is executed, e provides the current element and its location *(i,j)* within the table (see Figure 1). At the first call (highest level of recursion) e is the root element of the serializable information model S and *(i,j)=(1,1)*. The cells of the sub-elements of the current element e are recursively evolved until there is only one element with no further sub-elements left, i.e., a leaf.

The following section describes how an element (i.e., T_S-cell) of the information model in SOF (as suggested by the function *fill_T-Cell()*) has to be coded.

```
createStructuralComponent(Element e, T-cell (i,j))
 fill_T-cell(e, (i,j))
 for each sub-element c
  k=i
  i= createStructuralComponent(c, (i,j+1))
  connect all T-cells between (k,j) and (i,j) to one Ts-cell
  if no sub-elements exists
   return i+1
  else
    return i
```

Figure 1. Algorithm for the construction of T_S.

2) Syntax and Markup for the T_S-cells

The name and specifications regarding the cardinality and data types written in the *T*-cell are predetermined by the information model S. The syntax markup depicted in Figure 2 has been developed in order to represent each element's features.

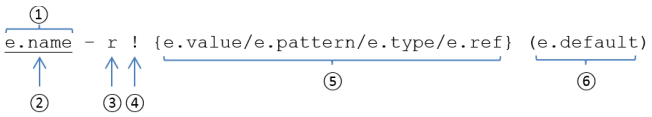


Figure 2. Syntax of the cell markup.

a) Name of element:

① The name of the element *e*.

b) Markup concerning the cardinalities:

② If *e* is a *required* element, ① will be underlined.

④ If *e* is an *at-most-once* element, it will be marked with "*!*".

→ If *e* is a *prohibited* element, see ⑤.

→ If *e* is an *exactly-once* element, ① will be underlined and marked with "*!*".

→ If neither the name of *e* is underlined nor the annotation "*!*" is used, *e* is an *optional-many* element.

③ Restrictions regarding the cardinality and data types of *e* that are determined by other elements are defined as rules in the rule component. Each rule of *r* is referenced using unique Roman numerals (e.g., "*III*") with an optional prefix or universal rules (see Section B.2).

c) Markup concerning data types:

⑤ The value range of *e* is illustrated using curly brackets. For instance, within an XML-based standard the value range is interpreted using the following XML Schematron definitions: *value, pattern, type, ref*. In the case of *ref* anon, the value range of the referencing XSD element is used.

→ If *e* is a *prohibited* element, "*Ø*" will be annotated instead of the value range.

⑥ If *e* holds a *default* value, it will be noted within round brackets.

Figure 3 shows an extract of the SAML specification [6], which will serve as a continuous example in this report.

Figure 4 shows how the structure component is applied to that SAML extract.

```
1. <?xml version="1.0" encoding="US-ASCII"?>
2. <schema targetNamespace="urn:oasis:names:tc:SAML:2.0:assertion"
  xmlns=http://www.w3.org/2001/XMLSchema
  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" version="2.0">
3.  <!–[...]-->
4.  <element name="Assertion" type="saml:AssertionType"/>
5.  <complexType name="AssertionType">
6.   <sequence>
7.   <element ref="saml:Subject" minOccurs="0"/>
8.   </sequence>
9.   <!–[...]-->
10.  </complexType>
11.  <element name="Subject" type="saml:SubjectType"/>
12.   <complexType name="SubjectType">
13.   <!--[...]-->
14.       <element ref="saml:SubjectConfirmation"
  maxOccurs="unbounded"/>
15.   </complexType>
16.  <element name="SubjectConfirmation"
  type="saml:SubjectConfirmationType"/>
17.   <complexType name="SubjectConfirmationType">
18.     <sequence>
19.       <!--[...]-->
20.       <element ref="saml:SubjectConfirmationData"
  minOccurs="0"/>
21.     </sequence>
22.     <attribute name="Method" type="anyURI" use="required"/>
23.   </complexType>
24.  <element name="SubjectConfirmationData"
  type="saml:SubjectConfirmationDataType"/>
25.   <complexType name="SubjectConfirmationDataType" mixed="true">
26.     <complexContent>
27.       <restriction base="anyType">
28.         <attribute name="NotBefore" type="dateTime"
  use="optional"/>
29.         <attribute name="NotOnOrAfter" type="dateTime"
  use="optional"/>
30.         <attribute name="Recipient" type="anyURI"
  use="optional"/>
31.         <!--[...]-->
32.       </restriction>
33.     </complexContent>
34.   </complexType>
35.  </schema>
```

Figure 3. Extract from the XML Schema Definition of a SAML assertion.

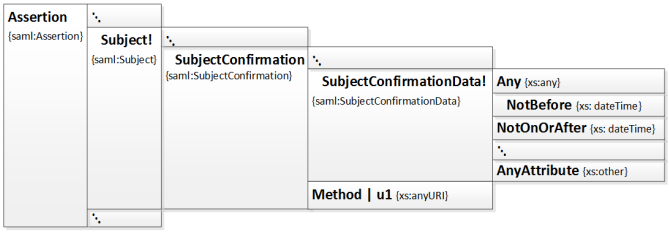


Figure 4. Extract of a SAML assertion as a structure component in SOF.

3) Referencing and Navigation using Identifiers

Each serializable information model *S* can be transformed into a tree. In order to refer to an element within such a tree, it is sufficient to provide, for instance, the path in a tuple form $(x_1, x_2, \dots, x_n)$. The path has to be followed from the root element in order to get to the last element. After the construction of T_S, a T_S-cell is accessible by a special navigation through a tuple referring to the same element (see algorithm in Figure 5). If $(x_1, x_2, \dots, x_n)$ is a tuple, describing an element *e* within the tree of the information model *S*, then the navigation in T_S will be, as shown in Table III.

Figure 6 applies the algorithm on the sample SAML extract (compare to listing in Figure 3).

```
1. while i < n
2.  goto x_i − 1 cell downward
3.  goto the top right-hand column relative to the current cell
4. goto x_i − 1 cell downward and print output
```

Figure 5. Algorithm to navigate within the tree of the information model.

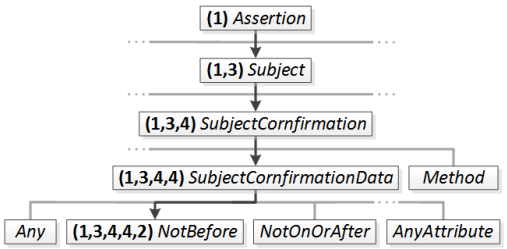


Figure 6. Navigation within the tree representation for an SAML assertion.

The table-oriented navigation within the structure component, according to the path in Figure 6, is further illustrated in Figure 7.

To keep the rules short, elements of *S* are referenced with tuples instead of their full names.

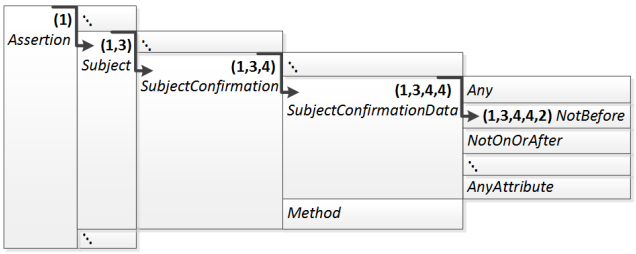


Figure 7. Navigation path within the structure component.

B. The Rule Component

The rule component for an information model *S* holds rules that *S* has to fulfill at all times, i.e., evaluates to *true*. Each rule is identified by unique Roman numerals to reference it within the T_S-cells of the structure component. Vice versa the T_S-cells addressed within a rule are identified by their tuple identifier.

1) Syntax of the Rule Component:

The syntax of the rule component is defined using a Domain-Specific Language (DSL) created in Xtext [7]. The syntax of the DSL will be explained in the subsequent paragraphs. The basic elements are sets that are categorized as follows:

a) Simple Sets

- *Enumerative sets* specify the elements contained (via their identifiers), separated by commas and encompassed by curly brackets.
- *Defined sets* are an accumulation of elements that fulfill specific characteristics regarding their cardinality. The four defined sets are A_S, R_S, P_S and I_S:
  - A_S contains all *at-most-once* elements,
  - R_S contains all *required* elements,
  - P_S contains all *prohibited* elements and
  - I_S instanced in the instantiation of *S*.

Therefore, the *defined sets* are S-specific and defined within the structure component's context since its elements are already underlined and marked with the respective symbols ("*!*" and "*Ø*").

b) Feature Sets

Feature sets are denoted as "*[A]*" and contain the elements that fulfill the requirements of statement *A*. For example, the set $[\{(1),(1,2)\} \subset \{(1),(1,3)\}]$ evaluates to the set $\{(1)\}$ since $[\{(1)\} \subset \{(1),(1,3)\}]$ is *true,* but $[\{(1,2)\} \subset \{(1),(1,3)\}]$ is *false.*

The sets C and D can be combined with the following operations and relations:

- $C - D$ forms the set of all elements of C except those contained in D.
- $C + D$ forms the set of all elements contained in C or D.
- $C \cap D$ forms the set of elements that are contained in C and D; machine-understandable: C *intersect* D.
- $< x >$ provides the value of the element with the identifier x.

Statements are created through the subsets' correlations or the set operators, respectively:

- $C \subset D$ is true if and only if all elements contained in C are also contained in D; machine-understandable: C *subsetOf* D.
- $C \not\subset D$ is true if and only if all elements in C are not contained in D; machine-understandable: C *notSubsetOf* D.
- $\#C$ provides the number of elements in C. A statement is formed with the #-operator, the relational operators >=, <=, =, >, < together with an accompanying integer.

The statements can be linked with the known sentential connectives AND, OR, XOR, and =>.

Operator and sentential connective ranking order: As there is no existing operator and sentential connective ranking order, the latter has to be defined using appropriate brackets so that the nesting represents the desired priority.

 ISBN: 978-1-61208-367-4

When using SOF, it became apparent that there often was a repetition of rules with the same content. To avoid the latter, so-called universal rules have been established.

2) Universal Rules

A universal rule *u* applies to each T_S-cell in which *u* has been referenced. Subsequently, all universal rules are semantically limited in their cardinality since the referenced elements of the serialized information model *S*, which are underlying the universal rules, have to be derived from the position of the annotated field. An example of such a universal rule is *uI*:

$$uI\colon \left(\left\{\left(x_1, \dots, x_{\#(x)-1}\right)\right\} \subset I_S \; AND \; \{x\} \subset R_S\right) XOR$$

$$\left(\left\{\left(x_1, \dots, x_{\#(x)-1}\right)\right\} \not\subset I_S \; AND \; \{x\} \not\subset I_S\right).$$

The rule states that the element *e* annotated with *uI* has to be instantiated if and only if its parent element has been instantiated.

C. Example of a Validation

This section briefly demonstrates how a validation is conducted by validating, if a profile conforms to a standard. In order to make a correct statement to this effect, the problem will be reduced to three subset problems. Assume a profile *P*, represented in SOF, which is derived from an existing standard *S* that itself is represented in SOF.

P is a valid profile of S if and only if the following subset relations are fulfilled, so that $P \subset S$ evaluates to *true*.

a) For the A, R, P sets of the respective rule component (of P and S) that represent the cardinalities of all elements: $A_P \subset A_S$, $R_P \subset R_S$, $P_P \subset P_S$.

b) Let D_{X_i} be the set-representation of the data type or the value of the model element i of the model X, as defined in Section A.2). The following must apply for each element (cell of the structure component) with the identifier i:

$$\forall i \in P\colon D_{P_i} \subset D_{S_i}.$$

c) Assume *S* is consistent, i.e., there are no rules that contradict each other:

$$\bigcap_{r=I,II,\dots} r \neq \{\}.$$

Then, each rule $r = I, II, \dots$ of the rule component needs to be checked whether *P* is included in the set interpretation of this rule:

$$\forall r = I, II, \dots : P \in r \Leftrightarrow P \in \bigcap_{r=I,II,\dots} r\,.$$

IV. TRANSFORMATION FROM SOF TO XML

To realize a platform-specific binding, a data model being developed using SOF can be transformed into an XML representation. It appears inconvenient to define structural properties within Schematron, so it will be assumed that the structure is defined using an XML Schema that is acting as the structure component. The semantic restrictions are defined by embedded Schematron rules that are acting as the rule component. The XPath expressions of the *rule* elements *context*, *assert* and *report* are evaluated to a Boolean expression, while its constructs can be translated, as shown in Table II.

To exemplify how the transformation works, a rule defined in SAML Profiles [8] is displayed both in Schematron and in SOF. The rule states that if an attribute *Method* is given the URI-value "*urn:oasis:names:tc:SAML:2.0:cm:holder-of-key*", then "One or more *<ds:KeyInfo>* elements MUST be present within the *<SubjectConfirmationData>* element. An *xsi:type* attribute MAY be present in the *<SubjectConfirmationData>* element and, if present, MUST be set to *saml:KeyInfoConfirmationDataType* (the namespace prefix is arbitrary but must reference the SAML assertion namespace)" [8].

It is impossible to express this rule exclusively using XML Schema. A schema validation check would be insufficient, so a Schematron rule (see Figure 8) is embedded within an XML Schema for the SAML example.

TABLE II. EQUIVALENTS OF XPATH AND SOF

Language construct	**Language**	
	XPath	***SOF***
Path	a/b/c	(1,2,3)
Structure	<Rule context = „A“> <assert a=“B“> </assert>	A => B
	[a::b = "predicate"]	[<(1,2,3)> = "predicate"]
Operators/ Relations		
Integer	+, -, *, /, <, <=, =, >=, >, !=	
Boolean	and, or	
String	= , !=	
	concat()	++
Node set	count()	#
Node	string()	<>

```
1. <pattern id="subject-confirmation">
2.  <title>Holder of Key</title>
3.  <rule context="saml:SubjectConfirmation[@Method =
    'urn:oasis:names:tc:SAML:2.0:cm:holder-of-key']">
4.   <assert test= "saml:SubjectConfirmationData/ds:KeyInfo">
5.     Message1
6.   </assert>
7.   <assert test= "not(saml:SubjectConfirmationData[@xsi:type]) or
     saml:SubjectConfirmationData[@xsi:type =
     'saml:KeyInfoConfirmationDataType']">
8.     Message2
9.   </assert>
10.  </rule>
11. </pattern>
```

Figure 8. Exemplary Schematron rule for SAML profiles.

Using SOF, the same rule within the rule component in combination with the respective structure component, as depicted in Figure 9, is defined.

```
   //Holder of key
1. <(1,3,4,5)> = "urn:oasis:names:tc:SAML:2.0:cm:holder-of-key" =>
2. ({(1,3,4,4,1)}subsetOf R AND <(1,3,4,4,1)> subsetOf "ds:KeyInfo"
3.  AND ({(1,3,4,4,01)} subsetOf I => <(1,3,4,4,01)> =
  "saml:KeyInfoConfirmationDataType"))
```

Figure 9. Rule for SAML profiles represented in SOF.

V. CONCLUSION AND FUTURE WORK

A Set-Oriented Formalism (SOF) consisting of a structure component and rule component has been introduced in this work-in-progress report. The SOF defines a machine-understandable formalism for the specification of data models on a logical level (as part of information exchange specifications). The defined data models are fully formalized and, therefore, machine-understandable, allowing them to be verified automatically.

The SOF acts as the foundation for an underlying methodology that aims to support modelers/developers in creating complex information exchange profiles based upon a set of data models and process models. The goal is to represent the models and relations between the models in a fully formalized notation so that integrity and verification checks can be automatically performed and platform-specific bindings can be generated automatically. No further usage of different notations/standards for the specification, profiling, and binding is needed, as all of those steps are covered by a single formalism.

The current development state of the SOF allows to define data models and rules and to verify a single data model. Modeling of process models is not yet available. Further research is needed to identify whether an algebraic calculus is suited to cover the needed requirements for defining and verifying process models and the relations between models. In addition, it is already possible to derive a profile from a data model. A graphical user interface is planned to make those steps easier to use. Implementing a verification component is planned to verify a given information model to the syntax compliance as well as the semantic correctness with respect to the underlying rule component. Furthermore, components for the automatic generation of platform-specific binding are intended (XML Schema and Schematron as well as HL7 FHIR [9]). Finally, the formalism needs to be extended to work with a set of models so that relations (requirements, dependencies, rules, constraints, pre-/post-conditions) between process steps can be defined and verified.

Subsequent papers and publications are planned that will describe further components around the SOF (such as process modeling, combination of formalized specifications, multi-model verification, etc.).

REFERENCES

[1] M. Belaunde et al., “MDA Guide Version 1.0.1,” June, 2003.

[2] “Unified Modeling Language (UML).” [Online]. Available: http://uml.org/. [retrieved: August, 2014].

[3] “OCL.” [Online]. Available: http://www.omg.org/spec/OCL/. [retrieved: August, 2014].

[4] W3C, “W3C XML Schema.” [Online]. Available: http://www.w3.org/XML/Schema#dev. [retrieved: August, 2014].

[5] ISO/IEC, “Schematron.” [Online]. Available: http://www.schematron.com/. [retrieved: August, 2014].

[6] S. Cantor, J. Kemp, R. Philpott, and E. Maler, “Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf.” März-2005.

[7] “Xtext.” [Online]. Available: http://www.eclipse.org/Xtext/. [retrieved: August, 2014].

[8] J. Hughes et al., “Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0, http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf.” March, 2005.

[9] HL7, “HL7 FHIR.” [Online]. Available: http://www.hl7.org/implement/standards/fhir/. [retrieved: August, 2014].

 ISBN: 978-1-61208-367-4

Towards Automated Design Smell Detection

A Proof of Concept in Detecting Opportunities for the Strategy Design Pattern

Stefan Burger
Software Engineering Group
University of Mannheim
Mannheim, Germany
sburger@mail.uni-mannheim.de

Oliver Hummel
Software Design and Quality
Karlsruhe Institute of Technology
Karlsruhe, Germany
hummel@kit.edu

Abstract— **Patterns are widely seen as an important ingredient to improve structure and maintainability of object-oriented software designs. In order to fully recognize opportunities for them, however, developers usually need a lot of experience as well as a good understanding of a given system. Hence, they often miss possibilities to use design patterns and produce code containing "design smells". With a view to overcome this unsatisfying situation, we have derived predicates that allow automatically identifying those locations in software systems where the Strategy design pattern would be beneficial. Moreover, we have implemented a prototypical tool that is able to apply these predicates. Using it on eight open-source projects with roughly 850K lines of code as an explorative study has discovered a variety of places where the pattern would improve the design. As ongoing work has demonstrated that this approach is transferable to other patterns, we believe that it has a good potential to increase the use of design patterns and therewith code quality in the not too distant future.**

Keywords-Design Patterns; Pattern Recommendation; Strategy; Code Quality

I. INTRODUCTION

Creating a clean and comprehensible design is probably one of the most challenging aspects in the development of complex software systems [1]. Hence, it does not surprise that it usually requires a lot of time and experience until software engineers have mastered all subtleties involved therein. In order to mitigate this steep learning curve, the object-oriented development community has collected a comprehensive set of so-called design patterns over the last decades. The best known pattern compilation is probably the seminal book of Gamma et al. (the "Gang of Four" (GoF), [2]) that lists 23 of them. However, since patterns are merely abstract solutions for common problems, they need to be tailored to a given context and consequently, applying the right pattern in a concrete situation is already a challenge in itself.

In order to break out of this vicious circle, the support of a (potentially proactive) recommendation system [3] that is able to recognize and suggest opportunities for the use of design patterns directly in common programming environment certainly seems like a promising idea. In recent years, numerous recommendation engines have been developed, including tools intended to simplify the usage of complex application programming interface (API) [4] or generally aiming on increasing the amount of reuse in software development [5][6]. However, despite the popularity of design patterns, there have only been few attempts to automate the detection of existing patterns in source code (such as [7]). Obviously, the idea of detecting pattern opportunities is remotely related with works on smell detection in the context of refactoring (such as by van Emden et al. [9]). Nevertheless, pattern recommendation requires an "understanding" for larger source ensembles that is usually not necessary for the relatively fine granular refactorings collected in Fowler's well-known book [10]. One important work on pattern recommendation has been presented by Briand et al. [11]. The authors presented a semi-automated decision support system intended to help developers find places for the use of patterns in Unified Modeling Language (UML) design diagrams and proved its feasibility for one pattern on a small case study with 15 classes. [8]. To the best of our knowledge, the only approach that directly aimed at automatically recommending promising "hot spots" in the code for the use of design patterns so far was recently presented by Christopoulou et al. [25]. We will discuss this and other related work in more detail in Section II.

Hence, the fully automatic approach for the detection of "design smells" and prospective design patterns based on static code analysis we describe in this paper is entering a largely unexplored territory. The most obvious benefits of such a pattern recommendation system are its support for novice developers who want to learn about good design in order to enhance the structure of their code. Moreover, it would also disburden experienced colleagues, for whom the recognition of pattern opportunities often still remains a challenging cognitive task, even after decades of experience [12]. Finally, such a system could also be used to get a new impression on code quality, as it would allow judging whether a system is well structured or still bears improvement potential in terms overlooked pattern opportunities.

In order to explain our pattern recommendation approach, we start by briefly discussing related work on design patterns and refactoring recommendations in Section II. In Section III, we exemplarily explain our pattern recommendation approach with the Strategy pattern, before we discuss how meaningful thresholds for the used metrics can be found in Section IV. The section following thereafter briefly describes the prototypical tool we have developed and its application on eight open source systems with 850 thousand links of cod (KLOC), before we conclude our paper with an outlook on future work and a summary of its contributions.

 ISBN: 978-1-61208-367-4

II. Related Work

The general idea of supporting developers in the selection of design patterns in order to improve source code quality has been discussed in various publications. However, the degree of automation so far used to be low. One early approach that has been published by Palma et al. [14] proposes the use of an expert system. It is based on the Goal-Question-Metric (GQM) method and uses a specific question template for every pattern. A developer can go through these templates in order to find the best matching pattern for a specific situation. However, this approach is completely manual and independent from the actual source code. Durdik et al. [12] have also been working on a set of questions intended to help documenting decisions for design patterns in order to facilitate replicability and hence program maintenance and evolution. A different approach was presented by Suresh et al. [15] who were using information about pattern usage (motivation, consequences, etc.) from other developers to create a pattern recommendation system. Again, the recommendation is based on disruptive questioning about a given situation and has no direct connection to the source code. Briand et al. [11] have proposed a similar semi-automatic approach that uses decision trees to identify places where GoF patterns might be useful within UML designs. Since not all necessary information can be derived automatically by this system, the developer needs to answer questions there as well. Moreover, their approach, supporting seven patterns in total, needs a comprehensive set of UML design diagrams that is often not available in practice. To our knowledge, the only approach similar to our work was recently published by Christopoulou et al. [25]. Their work also focusses on identifying Strategy pattern candidates, however they merely use an analysis of conditional statements without analyzing the surrounding method or class. Moreover, they do not give any rationale when it is worthwhile to recommend a pattern.

While such pattern recommendation is a relatively new research strand, automatically identifying potential code smells and related refactorings have been researched to some extend in recent years: As an example, consider Seng et al. [16] who have utilized software metrics in order to detect code smells and therewith identify potential places for code refactorings. However, the recommendations generated by their tool tended to break higher level structures such as design patterns. Hegedűs et al. [17] aimed to connect the usage of design patterns with software maintainability. They measured several hundred revisions of the open source project JHotDraw [27]. During their analyses, they found evidence that patterns can improve source code quality. Huston [18] analyzed the effects of design patterns on applications and their metrics scores. He developed a mathematical model based on software metrics (such as Coupling between Objects) to compare source code with a pattern and the same code without a pattern. His conclusion was that patterns can reduce high metric scores, but the usage of software metrics seems generally questionable in this context. This conclusion is also supported by Burger and Hummel's work that showed that refactorings often worsen metric values. Tourw'e et al. [28] have been working towards detecting refactoring opportunities or, in other words, code smells [10]. They are using logic meta-programming (LMP) for identifying smelly structures in the source code and for choosing an appropriate refactoring.

Another interesting challenge is identifying already implemented patterns in a given source code to be able to assess whether they have been applied in a meaningful way. The pattern detection community, e.g., comprising researchers like Baranski et al. [6] and others has been tackling this challenge for several years and has reached significant results, i.e., they have created pattern detection tools using various different technologies and approaches. Heuzeroth et al. [8] use static analysis of the source code for this purpose. Guéhéneuc et al. [20] have developed a combined approach based on a numerical signature (e.g. size/complexity, number of methods/parents, etc.) and a structural analysis of code files to identify design patterns. Tsantalis et al. [7] have proposed an approach which uses graph algorithms for identifying potentially modified design patterns. Fabry et al. [19] have developed an approach for detecting existing patterns, which is independent from the used programming language through extracting meta-information, such as method calls or variable references from the parse tree for this purpose.

III. Pattern Recommendation

This section explains our generic approach for a fully automated recommendation of design patterns and the necessary steps for detecting concrete candidates (we will use the abbreviation DPC for "design pattern candidate" in the following) in a source code, exemplarily using the GoF Strategy pattern to illustrate it. Step one of our approach is deriving the abstract syntax tree (AST) of a given Java source code, i.e., usually one .java file. Step two is extracting the necessary information (metrics and structural information) from the AST as a base for identifying DPCs. For this purpose, we aim to create a predicate for each supported pattern (to be explained in the upcoming subsections) that helps in recognizing the candidates. A DPC is found whenever all metric thresholds of a predicate are triggered by the underlying source code, or in other words, whenever the predicate evaluates to true. A graphical summary of this process is presented in Figure 1 .

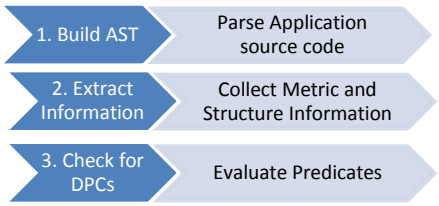


Figure 1 Pattern Candidates Identification Process.

Based on this model, we exemplarily describe the predicates we have defined for the Strategy pattern (see Figure 2) in the following subsections in more detail. According to Gamma et al. [2] the Strategy pattern is defined as follows: "*Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.*" Following this defini-

 ISBN: 978-1-61208-367-4

tion, the important part of the Strategy design pattern is the separation of different algorithmic strategies from the context in order to better support the open/closed principle, which states that code should be open for extension without the need for modification [21]. Thus, the strategies are implemented independently in separated classes that each gives home to a "family" of different algorithms (i.e. the methods A and B in Figure 2). Obviously, they all need to implement the common IStrategy interface that defines which methods should be available.

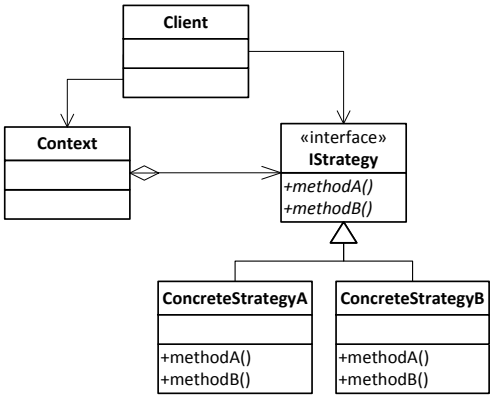


Figure 2 Class diagram for the Strategy pattern.

Although the GoF book and other literature (as e.g. [12]) provide some general guidance when to use a pattern, all are relatively imprecise when it comes to concrete rules for actually using a pattern. For an automatic recommendation system, however, it is obviously essential to define precise detection rules with good thresholds so that a suggested pattern is helpful and does not induce more complexity than it actually resolves.

The predicate for the Strategy pattern differs only slightly from the one for the State pattern, which we have also started to investigate. This is no surprise, since both patterns are aiming at encapsulating program behavior in separate classes in order to make it more exchangeable. Hence, candidates for both can be detected within large conditional (if/switch) statements depending on the same variable. The central difference of the two is conceptual: states typically "decide themselves" when an object should switch into another state in order to behave differently. For Strategy, this decision is triggered by an external event, such as a decision of the developer or the user of a system so that no object variable should be changed in the body of the conditional. Thus, the predicate for the Strategy pattern can be written as in Table I.

TABLE I. PREDICATE FOR STRATEGY CANDIDATES.

Rule	Description
R1.1	*In serveral methods of a class there exists an if/switch statement, which has a similar number of cases and uses the same attribut*
OR	*or parameter in the condition.*
R1.2	*In a class hierarchy there exists a number of subclasses, which are all overriding the same method(s) of the super class.*

Each rule of the predicate aims at identifying a different design smell indicating a possibility to use the Strategy pattern and is composed of a number of metrics based on code characteristics like number of subclasses or common attributes. If all metrics of a rule are passing a predefined threshold, a smell has been identified. Table II describes the metrics defined for the rules R1.1 and R1.2.

TABLE II. METRICS FOR STRATEGY DETECTION RULES.

No.	Metric	Type	Rule
M1.1	*Number of methods containing a conditional statement*	*Numeric*	*R1.1*
M1.2	*All methods of M1.1 are in the same class*	*Boolean*	*R1.1*
M1.3	*Every method identified in M1.1 has a conditional with an identical number of cases*	*Boolean*	*R1.1*
M1.4	*There exists a common attribute / parameter used in all cases of M1.1*	*Boolean*	*R1.1*
M2.1	*Common super class*	*Boolean*	*R1.2*
M2.2	*Number of overridden methods*	*Numeric*	*R1.2*
M2.3	*Number of subclasses overriding the same method*	*Numeric*	*R1.2*

In order to avoid choosing "arbitrary" thresholds, we have chosen them based on a careful analysis of numerous Strategy implementations retrieved from the Merobase software search engine [23], as explained in the next section.

IV. THRESHOLD DEFINITION

One of the most critical aspects for the acceptance of our envisaged approach is determining the thresholds that trigger a recommendation. Only with meaningful thresholds, it is possible to decide if a metric result indicates a design smell that should be resolved through the use of a design pattern or not. Thus, in this section, we explain exemplarily how we have derived the thresholds for the Strategy pattern. It should nevertheless be obvious that this procedure can also be used for the analysis of other patterns. The basic process contains four different steps, beginning with identifying the needed characteristics of the target pattern, i.e., the rules that might indicate the use of a pattern (cf. Table II). In order to establish grounded thresholds for a pattern, we considered an empirical analysis of existing pattern uses as the best solution so that the second step aims at identifying them with the help of a code search engine. The next step then is to measure the characteristics defined in step one for the discovered pattern instances. Finally, the thresholds can be derived from the measured values through a statistical analysis. Figure 3 illustrates the overall process graphically.

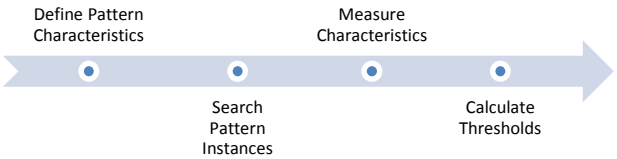


Figure 3 Finding meaningful thresholds for pattern recommendation.

The definition of the pattern characteristics in step one can be analytically derived from the explanation of a pattern, as described in the last section. Spotting concrete pattern instances in source code, however, as needed for the second step, is still an area of active research (e.g. [20]) without any tools that would be readily usable "off the shelf". Since we

 ISBN: 978-1-61208-367-4

nevertheless needed to come up with a way for finding a serious number of pattern instances with a reasonable amount of effort, we decided to use a software search engine. Since we are not aware of any search engine that would recognize patterns based on their structure, we needed to "guess" names that could be used as search terms. Fortunately, the Merobase [5] search engine allows to search for wildcards under a certain constraint, which is that the asterisk as wildcard character cannot be used as the first or the last character of a search. Under the assumptions that many Java programmers start interfaces with a capital 'I', as, e.g., suggested by Beck [22] and that the pattern name will also be reflected in the interface, we derived the following query:

*I*Strategy lang:java type:interface (protocol:svn OR protocol:CVS) original:yes*

Moreover, as is visible in the query, Merobase is able to limit searches on a desired programming language (here: Java), and a certain file type (i.e. interfaces). Moreover, we limited our analyses on Subversion (SVN) and Concurrent Version System (CVS) repositories as we assumed to find more mature projects there than in results from the open web and excluded identical duplicates. Thus, the delivered results contain every Java interface that starts with an 'I' and ends on Strategy. Merobase finds something over 250 matches for this query. We have analyzed the first 50 projects of the result set with a maximum of three patterns from one project in order not to bias the results towards the habits of a specific project. Moreover, we filtered out about 33 obviously "incorrect" implementations that did not comply with the recommendations of the Gang of Four [2] (e.g. they were just implementing a single Strategy) so that a total of 68 Strategy implementations remained.

The histogram in Figure 4 illustrates the size of Strategy implementations on the x-axis, i.e., how many subclasses of the Strategy interface or methods the analyzed instances of the pattern contained. The y-axis shows how often each case has occurred during the analysis.

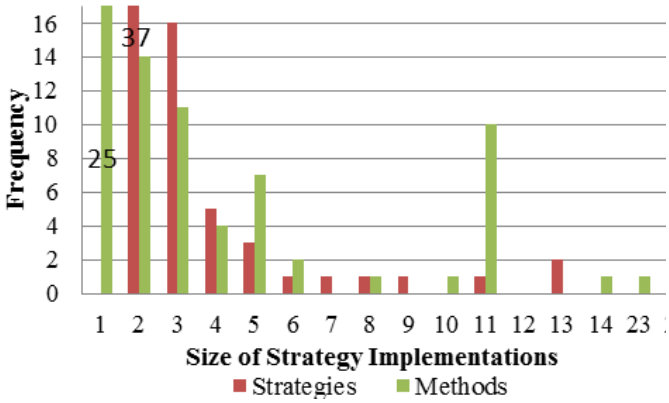


Figure 4 Distribution histogram of strategies and methods.

A statistical overview of the analysis results is shown in Table III.

TABLE III. STATISTICAL FACTS OF THE MEASUREMENTS.

	Strategies	*Methods*
Minimum	2.00	1.00
Median	2.00	2.00
Average	3.29	3.37
Maximum	13.00	25.00

As mentioned before, we merely considered Strategy patterns containing 2 or more concrete strategies. On the other hand, interfaces with only one method in at least two strategies were included in the results.

As the statistical analysis has revealed, existing Strategy implementations are quite different so that it is hard to come up with fixed threshold values. Hence, we decided to create a staged recommendation model based on the average and median results. Although M1 and M2 (cf. Table II TABLE II. aim at identifying strategies "hidden" in the code in a different way, the same thresholds can be applied since both are based upon the number of strategies and the number of implemented methods per Strategy. Therefore, our model illustrated in Figure 5 assigns one of three levels of usefulness to each detected pattern recommendation as follows:

1. **P**ossible: a pattern is reasonable and it is likely that it could improve the code especially if further extensions are to be expected. A possible place for a Strategy is found in this case if the number of strategies and methods is at least equal to the median of the analysis presented in Table III, i.e., both values are at least 2.
2. **U**seful: a pattern is useful for an analyzed source code if the measurement results are at least 3, which roughly corresponds to the average of the analyses.
3. **R**ecommended: a pattern is definitively recommended when all measurements are over the average, i.e., if they are equal or larger than 4.

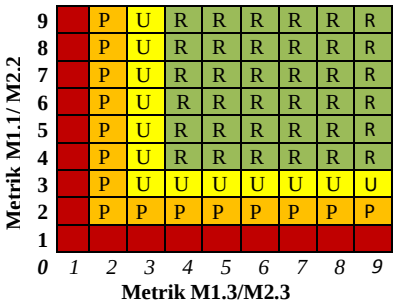


Figure 5 Graphical threshold model for Strategy smells.

Figure 5 illustrates the three levels of usefulness grpahically: Orange for **P**ossible, yellow for **U**seful und green for **R**ecommended.

V. DETECTION EXAMPLES

In this section, we demonstrate how our predicates can be used for automated design smell detection and pattern recommendation. We have analyzed eight open source programs with a total of about 850 thousand Lines of Code (KLOC) in 10,000 classes and found 41 candidates where the Strategy pattern was deemed helpful. Before we present the detailed numbers, we briefly explain the tool we have been developing for this purpose.

 ISBN: 978-1-61208-367-4

A. Detection Tool Preview

In order to evaluate the explained rules, metrics and thresholds automatically, we have implemented a detection tool based on PMD [24]. PMD is a code quality tool using the abstract syntax tree (AST) of Java for identifying code smells. Our tool is using the plugin interface of PMD in order to benefit from the PMD platform and avoid reinventing the wheel. It is able to extract the required data for the application of the rules defined in Table II from there and to finally present recommended patterns together with the measured values and of course the places (i.e. classes) where they should be integrated.

The code of the tool is separated into four components, respectively packages. The first package is collecting the necessary information from the AST. Package two implements the data model for storing the extracted information, while package three processes the data and measures the required metric values. After collecting all necessary information, the fourth package stores and evaluates the metrics as well as the structural information and finally applies the predicates to identify and present the potential pattern.

B. Detecion Results and Examples

As mentioned previously, we have chosen eight well-established open source projects for a first explorative study intended to illustrate the effectiveness of our approach and to help us gain a better understanding of its mechanics. Table IV provides an overview of all discovered Strategy design smells. Execution times were measured on an old 1-core computer with 2 GHz and can hence at least be divided by four on more recent machines. However, in order to provide acceptable times for a proactive recommendation system, applying an incremental analysis seems necessary.

TABLE IV. ANALYSIS RESULT OVERVIEW.

	jEdit	Lucene	Helix	Megamek	jHotdraw	Columba	tyrant	Jaffa
Possible	2	0	2	13	3	2	3	1
Useful	1	4	1	12	2	0	0	0
Recomm.	0	0	0	0	0	0	0	0
KLOC	117	90	35	283	80	91	41	113
Time (h)	1.5	1.1	0.6	18	1.9	3	1.4	3

Interestingly, no clear recommendation for the use of the Strategy pattern has been found, but a total of 45 occasions where the pattern at least appears to be possible. For the moment, we have manually inspected the discovered suggestions and consider them as appropriate. The next section on future work will discuss planned additional evaluations.

In order to illustrate the results more vividly, we have chosen a design smell discovered in the open source tool jEdit (Version 5.1) [29] as an example. The code snippet shown in Figure 6 was extracted from its *TextUtilities* class. It contains two methods (*findWordEnd*/*findWordStart*) with a switch statement in turn containing three cases using the same case condition (i.e. `WHITESPACE`, `SYMBOL` and `WORD_CHAR`) and the same switch parameter (`type`). Due to limited space, code details have been omitted.

```
public static int findWordStart(…) {
    switch(type) {
    case WHITESPACE:
        …
    case WORD_CHAR:
        …
    case SYMBOL:
        …
    } return 0;
}
```

```
public static int findWordEnd…) {
    switch(type) {
    case WHITESPACE:
        …
    case WORD_CHAR:
        …
    case SYMBOL:
        …
    } return line.length();
}
```

Figure 6 Examplary opportunity to use the Strategy pattern in jEdit.

Table Vsummarizes the assessment of the predicate defined in Tables I and II, respectively, for this example.

TABLE V. MEASUREMENT RESULTS FROM TEXTUTILITIES CLASS.

Metric	Value	Metric	Value
M1.1	2	M1.3	3
M1.2	True	M1.4	True

As visible in the Table V, both Boolean metrics (M1.2 and M14) are true and hence fulfill the first requirement for design smell detection. Moreover, M1.3 is equal to the average of 3 as well as M1.1 is equal to the median. According to the model described in Figure 5 , a Strategy pattern can be considered as a useful improvement for this piece of code.

VI. FUTURE WORK

We have planned to improve our prototypical recommender application so that it is able to detect and recommend pattern candidates for design patterns automatically for each Java project a developer is working on in a common integrated development environment. In this context, it is important to find a convincing way to present pattern recommendations to the users. A well-designed user interface that clearly indicates where a pattern could be introduced and which classes should participate in the pattern in what role is probably the key to achieving user acceptance. Another requirement is that it will most likely be necessary to extend the use of thresholds to the size of the code base, i.e. the size of code in the case blocks in case of the Strategy patterns. According to informal feedback of colleagues, it seems to be the case that developers are very sensitive when patterns create a relatively large overhead compared to the actual functionality they "contain". Moreover, we will continue working on recommendation predicates for further GoF patterns. We currently assume that we will be able to develop detection possibilities for most of the GoF patterns. Only prospective Adapter and Interpreter patterns cannot be suggested based on existing code since they require a cognitive decision of the developer to integrate a novel piece of code into an existing system. It is also impossible to recommend opportunities for the Composite pattern since a domain analysis has to detect the part-whole hierarchies between objects to be represented by this pattern.

Another important open topic is of course the evaluation and the fine tuning of the developed predicates and thresholds that we use for pattern candidate detection. We plan to analyze further open source projects in order to see whether

our tool is able to recommend appropriate pattern opportunities. In order to increase the validity of the results, we want to give the identified recommendations to various professional developers (or even the authors of the investigated systems themselves) in order to get an independent feedback whether they consider the discovered candidates as useful. Another validation we plan to tackle soon is scanning the repositories of open source projects for concrete refactorings that have integrated design patterns into their code base. Using our tool on the version before such a refactoring obviously should yield a recommendation for the appropriate pattern and further demonstrate the effectiveness of our approach.

VII. CONCLUSION

In this paper, we have presented a prototype of a design pattern recommendation tool that can be directly integrated into common development environments. It comprises the following three contributions. First, we have explained how opportunities for the use of design patterns can be identified through analyzing the AST of Java programs based on so-called detection predicates. Nevertheless, the presented ideas are not be limited to Java, but should be transferable to other object-oriented languages as well. Second, we have presented an approach on how meaningful thresholds for the metrics used in the detection predicate can be derived from mining existing Strategy implementations in open source projects.

Third, in order to demonstrate the practical feasibility of our ideas, we have presented concrete predicates for the GoF Strategy pattern as well as a concrete Java implementation for a detection utility and evaluated it on eight open source projects together comprising more than 850 thousand lines of code 10,000 classes. Our tool was able to present numerous meaningful opportunities for the utilization of the pattern. Hence, we are encouraged to continue our work in order to also define predicates for various other GoF patterns and extend our prototype accordingly.

REFERENCES

[1] C. Larman, Applying UML and Patterns, An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3/e, Pearson Education India, 2012.

[2] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design patterns: Abstraction and reuse of object-oriented design, Springer, 1993.

[3] M. Robillard, R. Walker and T. Zimmermann, "Recommendation Systems for Software Engineering", IEEE Software, Vol. 27, No. 4, pp. 80-86, 2010.

[4] R. Holmes, R. J. Walker and G. C. Murphy, "Strathcona example recommendation tool", ACM SIGSOFT Software Engineering Notes, pp 237-240, 2005.

[5] O. Hummel, W. Janjic and C. Atkinson, "Code Conjurer: Pulling reusable software out of thin air", IEEE Software, Vol. 25, No. 5, pp. 45-52, 2008.

[6] M. Baranski and J. Voss, "Genetic algorithm for pattern detection in NIALM systems", IEEE Intern. Conference on Systems, Man and Cybernetics, , pp. 3362-3468, 2004.

[7] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. Halkidis, "Design pattern detection using similarity scoring", IEEE Transactions on Software Engin., pp. 896-909, 2006.

[8] D. Heuzeroth, T. Holl, G. Hogstrom, and W. Lowe, "Automatic design pattern detection", IEEE International Workshop on Program Comprehension", pp. 94-103, 2003.

[9] E. Van Embend, and L. Moonen, "Java quality assurance by detecting code smells", Working Conference on Reverse Engineering, pp. 97-106, 2002.

[10] M. Fowler, Refactoring: improving the design of existing code, Addison-Wesley Professional, 1999.

[11] L. Briand, Y. Labiche, and A. Sauve, "Guiding the Application of Design Patterns based on UML Models", Intern. Conf. on Software Maintenance, pp. 234-243, 2006.

[12] K. Beck, et al., "Industrial experience with design patterns". Intern. Conf. on Software Engin., pp. 103 -114, 1996.

[13] Z. Durdik, and R. Reussner, "Approach for architectural design and modelling with documented design decisions", Int. Conf. on Quality of Software Architectures, p.9, 2012.

[14] F. Palma, H. Farzin, Y. Gueheneuc, and N. Moha, "Recommendation system for design patterns in software development", Intern. Workshop on Recommendation Systems for Software Engineering, pp. 49 – 54, 2012.

[15] S. Suresh, M. Naidu, S. A. Kiran, and P. Tathawade, "Design pattern recommendation system: a methodology, data model and algorithms", International Conference on Computational Techniques and Artificial Intelligenc, 2011.

[16] O. Seng, F. Simon, and T. Mohaupt, Code Quality Management, dpunkt Verlag, Heidelberg, 2006

[17] P. Hegedűs, D. Bán, R. Ferenc, and T. Gyimóthy, "Myth or Reality? Analyzing the Effect of Design Patterns on Software Maintainability", in Computer Applications for Software Engineering, Disaster Recovery, and Business Continuity, Springer, pp. 138 - 145, 2012.

[18] B. Huston, "The effects of design pattern application on metric scores", Journal of Systems and Software, Vol. 58, No. 3, pp. 261-269, 2001.

[19] J. Fabry and T. Mens, "Language-independent detection of object-oriented design patterns", Computer Languages, Systems & Structures, Vol. 30, No. 1, pp. 21-33, 2004.

[20] Y. Guéhéneuc, G. Jean-Yves and S. Houari. "Improving design-pattern identification: a new approach and an exploratory study." Software Quality Journal 18.1, pp. 145 - 174, 2010.

[21] R. Martin, Clean code: a handbook of agile software craftsmanship, Pearson Education, 2008.

[22] K. Beck,. Implementation Patterns. Pearson Education, 2007.

[23] W. Janjic, O. Hummel, M. Schumacher, and C.Atkinson, "An unabridged source code dataset for research in software reuse". Intern. Workshop on Mining Software Repositories, pp. 339 – 342, 2013.

[24] N. Rutar,C. Almazan, and J. Foster. "A comparison of bug finding tools for Java". In 15th International Symposium on Software Reliability Engineering, pp. 245 – 256, 2004.

[25] Christopoulou, A., Giakoumakis, E. A., Zafeiris, V. E., and Soukara, V., "Automated refactoring to the Strategy design pattern". Information and Software Technology, pp. 1202-1214, 2012.

[26] S. Burger and O. Hummel, "Über die Auswirkungen von Refactoring auf Softwaremetriken" (in German), GI-Fachtagung Softwaretechnik, pp. 113–126, 2012.

[27] http://www.jhotdraw.org/, JHotDraw as Open-Source Project, Accessed on 02/11/2014

[28] T. Tourw'e und T. Mens, „Identifying refactoring opportunities using logic meta programming," in Software Maintenance and Reengineering, 2003.

[29] http://www.jedit.org/, Progammer's Text Editor, Accessed on 02/11/2014

 ISBN: 978-1-61208-367-4

UCDMD: Use Case Driven Methodology Development

Hanieh Zakerifard, Raman Ramsin
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
e-mail: hzakeri@ce.sharif.edu, ramsin@sharif.edu

***Abstract*—Situational Method Engineering (SME) focuses on project-specific construction of methodologies based on the characteristics of the project situation at hand. Requirements Engineering (RE) is considered as a key activity in SME and is concerned with the elicitation, specification, modeling and validation of methodology requirements. However, unlike requirements engineering in software development, the RE methods currently practiced in SME are still immature, and methodology engineering has a lot to learn from Software Engineering (SE) in this regard. Use Cases are widely used in software engineering to express the functional requirements of software systems, and the use case model is an effective tool for capturing stakeholder requirements in a clear and unambiguous fashion. Despite its potential benefits, the use-case-based approach has not been used in SME yet. The main objective of this paper is to propose the UCDMD (Use-Case-Driven Methodology Development) methodology as a new object-oriented approach to SME; in this approach, methodology requirements are completely expressed in terms of use cases, and are utilized in a SME process for developing the target methodology. The use-case-driven nature of the proposed process promotes requirements traceability, and object-oriented realization of the use cases facilitates the implementation of CASE tools for the methodology produced.**

Keywords-situational method engineering; requirements engineering; use case modeling; use case-driven development

I. INTRODUCTION

When developing software systems, selecting the appropriate methodology is always an important issue. Nevertheless, after using software development methodologies for decades, developers have realized that there is no general-purpose methodology that suits every project situation. The need for project-specific methodologies has therefore resulted in the emergence of SME, which is specifically concerned with the construction/adaptation of a methodology according to the specific characteristics of the software development project at hand [1]. As in any development effort, it is important in SME to perform RE activities precisely, so as to ensure that the produced methodology satisfies the needs of the target software development project situation. RE in SME is concerned with eliciting, specifying and validating the real-world goals, functional/non-functional requirements, and constraints of a methodology in a specific project situation [2]. Although a wide range of RE approaches have been used in SE, the RE approaches which are used in SME are few and immature in comparison.

Use case modeling has become a popular technique for capturing and describing the functional requirements of software systems [3]. Use case driven SE approaches support requirements traceability during the development process, and assist in managing change and evolution [4]. As the use case model provides a high-level view of the interactions between the system and its users (actors), it has been effectively used for capturing the functional requirements of interactive systems. Use cases are vastly used in object-oriented software development methodologies [4], which prescribe various techniques for mapping use cases to their object-oriented software realizations.

A software development *methodology* is akin to a complicated interactive system in which interaction with the user plays a pivotal role: A methodology governs the software development process by prescribing the products that should be produced and the corresponding activities that should be performed, and it does all of this by providing guidance to its users, which mainly consist of managers, users, developers, and other project stakeholders. A SME effort is thus faced with the same problems and challenges which are encountered when developing any other type of interactive system; use cases are therefore potentially useful for elicitation and specification of methodology requirements in SME efforts. Furthermore, just as use cases are mapped to object-oriented software in software development methodologies, the use cases produced for methodology development can be mapped to custom-made software tools for enacting the target methodology. The target methodology can therefore be implemented as a methodology-based CASE tool; this makes the approach very appealing for use in a Process-centered Software Engineering Environment (PSEE). Despite their potential benefits, use cases have not been used for methodology development yet.

We propose UCDMD as a use-case-driven approach to SME in which requirements are expressed in terms of use cases, and the target methodology is developed through a process which prescribes the activities that should be performed and the products to be produced. Being use-case-driven means that all the artifacts of UCDMD are produced in order to realize the use cases; traceability is thus achieved.

The rest of this paper is organized as follows: Section II provides a brief review on the research background; Section III explains the proposed UCDMD methodology, and Section IV provides an example of its enactment; a criteria-based evaluation of the proposed methodology is presented in Section V; and Section VI provides the conclusions and suggests ways for furthering this research.

 ISBN: 978-1-61208-367-4

II. RESEARCH BACKGROUND

Although use cases have not been previously used as a basis for methodology development, they have been widely used in process modeling approaches; instances have been reported in [5][6][7][8][9]. However, this cannot be considered as use-case-driven SME.

In this section, the concepts and methods on which this research is based will be introduced. To this aim, we will first present an overview of RE in SME, and will then briefly introduce an existing process framework for SME; we have used this framework as the basis for developing UCDMD.

A. RE in SME

Since the advent of SME, different approaches have been proposed for RE in this context: The research reported by Ralyté [10] presents the roadmap-driven approach in which process-driven and intention-driven strategies are used for eliciting the requirements; a criteria-based approach has been proposed by Ramsin and Paige [11] in which requirements are identified through a top-down iterative-incremental process; and the framework proposed by Olsson et al. [12] is a comprehensive general process for RE in SME, providing detailed descriptions for the various activities and techniques prescribed. None of the above RE approaches is defined as part of a comprehensive SME process. In contrast, UCDMD is a comprehensive object-oriented SME process in which requirements (use cases) play a pivotal role in producing all the deliverables.

B. SME Process Framework

The generic pattern-based process framework for SME, proposed by Asadi and Ramsin [13], is made up of three serial *Phase* process patterns: Method Initiation, Method Construction, and Deployment (see Fig. 1). The phases of the framework consist of several *Stage* patterns along with their constituent *Task* patterns. The framework can be instantiated and configured to fit the SME situation at hand. We have used this framework for constructing UCDMD.

III. PROPOSED METHODOLOGY – UCDMD

In this section, our proposed UCDMD methodology will be described in detail. However, before delving into the particulars of UCDMD, we will first explain how the notion of use case has been adapted for application in SME.

A. Use Case Driven RE in SME

A use case represents a sequence of interactions between the system and its actors to achieve a specific functional goal of the system [14]. It is deeply rooted in the problem domain, and is understandable to all stakeholders. Use cases are prevalently used in SE. But in order to utilize them in SME, we should first devise a mapping between the notion of *use case* as used in SE to the notion of *use case* purposed for application in SME. In SME, the target product is a methodology, not a software system in the traditional sense of the term; methodology actors are the roles in the software development environment (e.g., developers and managers) which affect the methodology (e.g., by tuning it or providing it with information), or are affected (governed) by it; a methodology use case is an atomic *SE* activity or task which is prescribed and governed by the methodology and whose fulfilment is of value to at least one actor. A methodology's use cases are elicited from its users and can be based on the situational factors of the organization and the project at hand. However, as in SE, methodology use cases only capture the functionality expected from the methodology, not its nonfunctional features (such as seamlessness); furthermore, methodology use cases describe what a methodology does without specifying how it does it (in other words, methodology use cases are not concerned with *techniques*).

B. Levels of Modeling in UCDMD

Modeling is an integral part of any methodology. In SE methodologies, different levels of modeling are used for modeling the implementation-independent aspects of the system (problem domain) as separate from its implementation-specific features (solution domain). The same distinction is true in SME methodologies. However, there is no established definition for the problem and solution domains in the SME context. Therefore, the first step in developing a SME methodology is to define these domains and the different levels of modeling required (from Abstract to Concrete: Logical to Physical). We have used the levels proposed by Agh and Ramsin [15] as a basis for defining the following three modeling levels for UCDMD:

- *Methodology-Type-Independent Level*: This level signifies the problem domain in SME, focusing on the definition of *general* methodology requirements and features, regardless of methodology type (e.g., agile or plan-driven). Situational factors and use cases are modeled at this level, comprising the nonfunctional and functional requirements. General structural and behavioral modeling of the methodology is also performed, aiming at realizing the requirements by developing a general, type-less methodology.
- *Methodology-Type-Dependent and Technique-Independent Level*: At this level, the type of the target methodology is specified, requirements are realized based on the defined type, and relevant structural and behavioral models are produced/refined. Even though the type has been determined, the methodology only consists of activities and tasks which specify *what* should be done. This is because *techniques*, which describe *how* the activities and tasks should be performed, have been deliberately left out.
- *Technique-Dependent Level*: The techniques and technique-dependent elements of the methodology are added, requirements are realized based on these elements, and the relevant models are produced/refined.

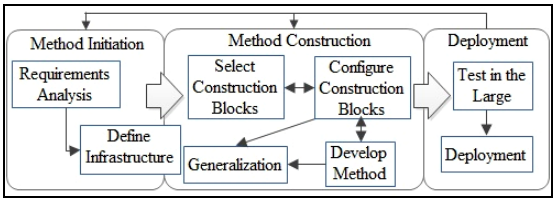


Figure 1. Generic SME Process Framework – Adapted from [13].

 ISBN: 978-1-61208-367-4

C. UCDMD Process

UCDMD consists of three serial phases, which in turn consist of iterative stages (see Fig. 2). The second phase is in fact UCDMD's iterative development engine. The phases and their internal stages will be explained in this subsection.

1) Initiation Phase

The objective of this phase is to provide a solid foundation for methodology construction. Eliciting and modeling the requirements and establishing the infrastructure of the target methodology are the main goals of this phase.

a) Requirements Engineering (Stage)

The aim is to define methodology requirements by eliciting, modeling, and prioritizing the situational factors and requirements. The activities are described below:

Capturing domain vocabulary: A glossary is produced of the main concepts of the problem domain. This document will help identify the actors, use cases and structural/behavioral elements of the target methodology.

Eliciting situational factors: Situational factors [16] are elicited through studying available documents and interviewing the users of the methodology (e.g., managers and developers). Documents may include organizational process documents, documents of the project at hand, and documents of the target methodology. The situation of the project is determined by giving values to the situational factors; these values will be updated based on the methodology type determined in the next phase. Lists of candidate situational factors are already available [17].

Mapping situational factors to functional/non-functional requirements: As situational factors are mainly non-functional in nature, they are mostly mapped to non-functional requirements of the target methodology. However, some situational factors can and will be mapped to specific functionalities of the target methodology; typical instances include situational factors which pertain to management issues, which are typically mapped to umbrella activities. These functional requirements will be documented to be used as candidate use cases after conflict resolution.

Resolving conflicts: In this stage, the conflicts that exist among the requirements are identified and resolved [17].

Identifying use cases: Starting from the initial list of functional requirements (mapped from situational factors), actors and use cases of the target methodology are identified through an iterative process. The process first focuses on identifying the actors (roles of methodology users); use cases are then identified/revised based on the expectations of the actors, resulting in a UML (Unified Modeling Language) use case diagram [3]. The question that should be asked from actors to identify their relevant use cases is: *"What are the software development activities that you expect the methodology to guide you through?"* The use cases thus identified are the SE activities on which the target methodology should provide instructions and guidelines. Use cases are therefore *constituents* of the target methodology.

Prioritizing use cases: Use cases are primarily prioritized based on business value, and then by the development risks involved. Use cases and their priorities are iteratively reviewed and revised during the development process.

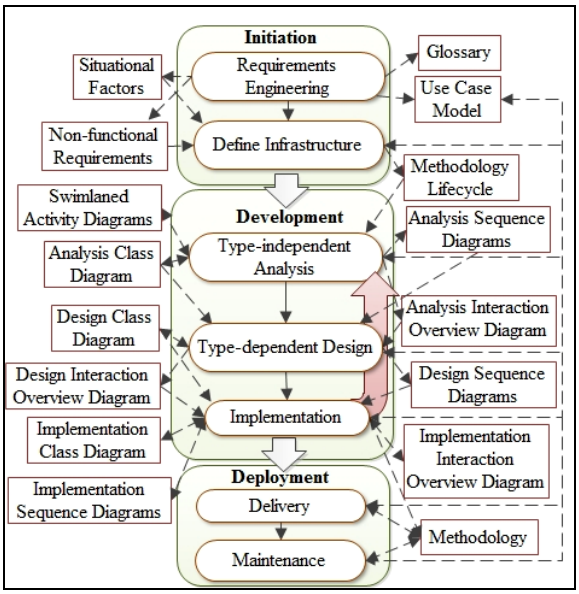


Figure 2. UCDMD Process.

Refining use cases: Detailed descriptions of the use cases are produced which elaborate on their preconditions, postconditions, actors, and flows of events (steps).

Structuring use case model: Structural relationships among use cases and actors (*generalization/specialization* and *include/extend*) are identified and added to the model.

Validating use case model: The use case model is verified and validated by methodology users. The checklist proposed by Cockburn [14] is very useful for this purpose.

b) Infrastructure Definition (Stage)

The objective of this stage is to determine the architecture of the methodology and acquire the required tools. The activities performed in this stage are as follows:

Establishing architecture: Based on the elicited requirements, a high-level lifecycle is defined for the methodology. This lifecycle is usually selected from among existing frameworks. If a specific lifecycle is not requested, the generic lifecycle proposed by Pressman [18] can be used.

Selecting tools: The tools (e.g., PSEE [16]) required for developing the methodology are identified and acquired.

2) Development Phase

The objective of this phase is to design and construct the target methodology. This phase consists of three stages which develop the methodology through an iterative-incremental process driven by the use cases.

a) Type-independent Analysis Stage

The aim of this stage is to produce structural/behavioral models for a general (type-independent) methodology which satisfies the use cases selected for realization in the current iteration. The activities of this stage are described below:

Structural modeling: Based on the use cases and non-functional requirements elicited, a UML class diagram is produced of the target methodology's structural elements. Existing frameworks, such as OPF (OPEN Process Framework) [19], can be used for identifying the classes. These analysis classes are of three general types: Work-units,

 ISBN: 978-1-61208-367-4

Products, and Roles (producers); however, many subclasses of each type are involved in constructing a methodology. Objects of these classes will interact to realize the use cases.

Behavioral modeling: Behavioral aspects of each use case are modeled in a UML activity diagram. The activity diagram is partitioned into swimlanes which correspond to the structural *objects* of the methodology (which realize the use case), and its actors in the context of the use case.

Realizing use cases: For each use case, the object interaction necessary for realizing the use case should be modeled in a UML sequence diagram. The swimlaned activity diagrams previously produced are used as bases for developing these analysis sequence diagrams.

Determining/Revising order of use cases: It is usually necessary for the use cases to be executed in a certain, predefined order. In this case, the order of execution is modeled in a UML interaction overview diagram.

Testing: The models produced in the current iteration are tested for completeness, accuracy, consistency, validity, and conformance to the methodology architecture.

b) Type-dependent Design Stage

The purpose of this stage is to develop a type-dependent and technique-independent version of the methodology, thus transitioning to the solution domain.

Determine Methodology Type (Sub-stage)

The aim of this sub-stage is to determine the type of the target methodology through the following activities:

Determining/Revising methodology type: If the type of the methodology has not been constrained by its users, it has to be determined based on the requirements. The type can connote the methodology's paradigm (e.g., object-oriented or agent-oriented), overall strategy (e.g., agile or plan-driven), design/implementation approach (e.g., component-based or service-oriented), application domain (knowledge-based or real-time), or a combination of the above.

Revising methodology infrastructure: The architecture of the methodology is refined based on the selected type. Instead of refining the current architecture, the methodology engineer may choose to replace it with an existing process framework. For instance, the Object-Oriented Software Process (OOSP) [20] can be used in case an object-oriented type is desired, and the framework proposed by Kouroshfar et al. [21] can be used if a component-based type is targeted.

Methodology-type-dependent Modeling (Sub-stage)

The objective of this sub-stage is to realize the use cases of the current iteration based on the methodology type, regardless of the techniques required for implementing the activities. The tasks of this sub-stage are described below:

Refining structural model: The analysis class diagram is refined and extended based on the methodology type, resulting in a *design* class diagram.

Realizing use cases (design): The use cases selected for the current iteration are realized based on analysis sequence diagrams, the design class diagram, and the new architecture; *design* sequence diagrams are thus produced.

Revising order of use cases (design): The interaction overview diagram is reviewed and revised based on the design sequence diagrams and the revised architecture.

Testing (design): Design models are tested for completeness, accuracy, consistency, validity, and conformance to the new architecture.

c) Implementation (Stage)

The methodology designed in the previous stage consists of activities which describe *what* is to be done, but falls short of specifying *how* the activities should actually be performed. The implementation stage is concerned with specifying the *techniques* which define how the activities of the methodology should be carried out. The target methodology is then constructed based on the specified techniques so that the use cases are satisfied.

Technique-dependent Modeling (Sub-stage)

The aim of this sub-stage is to determine techniques for implementing the target methodology's use cases. The activities performed in this sub-stage are described below:

Specifying techniques: Techniques are typically chosen from among those proposed by methodologies/frameworks which are of the same type as the target methodology; for instance, a list of agile techniques has been provided by Abad et al. [17]. Techniques are selected based on the use cases, non-functional requirements, and available resources.

Refining structural model (implementation): The structural model of the methodology (class diagram) is refined and extended based on the techniques introduced, resulting in the *implementation* class diagram.

Realizing use cases (implementation): Use cases are realized based on the design sequence diagrams, implementation class diagram, and the methodology so far produced, thus yielding *implementation* sequence diagrams.

Revising order of use cases (implementation): The interaction overview diagram is updated based on the added techniques. The resulting diagram is an extension of the design version, and should not contradict it in any way.

Method Construction (Sub-stage)

The classes which have so far been defined possess the final state and behavior necessary for realizing the use cases, and the sequence diagrams show how instances of specific classes should interact to realize the use cases. However, the final methodology should be configured from activities which correspond to the use cases, and which comprise a complete methodology that conforms to the defined architecture. The activities of this sub-stage are as follows:

Determining construction blocks: The structural elements that should be incorporated into the methodology in the current iteration are determined. By default, each use case is mapped to a coarse-grained construction block (activity). The structural elements (class instances) which should interact to realize the use case are also considered as construction blocks; these blocks are typically taken as internal elements of the activity corresponding to the use case. The method engineer can also choose to use method components retrieved from a repository.

Configuring construction blocks: The construction blocks defined in the previous activity are configured with appropriate preconditions/postconditions, and their internal structure is determined: The method engineer should decide which blocks should be incorporated into other blocks.

 ISBN: 978-1-61208-367-4

Integrating construction blocks into produced methodology: The construction blocks configured in the previous activity are integrated with the methodology built so far. The method engineer decides where each new construction block should go, and what changes should be made to facilitate the integration. It should be noted that multiple instances of the same block may be integrated into different phases/stages of the methodology.

Identifying reusable blocks: Reusable blocks of the methodology are identified and stored in a repository.

Testing: All products are tested for accuracy, consistency, validity, and conformance to the overall architecture.

Implementing supporting software: This activity produces software support for the methodology, in parallel with the development of the methodology itself. As previously observed, since an object-oriented use-case driven process has been followed for producing the methodology, the class diagrams and sequence diagrams produced can be directly used for implementing software support for the methodology (usually as a methodology-based CASE tool).

Reviewing iteration: Products, plans, and even the UCDMD process are reviewed and revised. Decision should be made to either initiate a new iteration (if unrealized use cases remain), or to proceed to deployment.

3) Deployment Phase

This phase aims to deliver the target methodology to its intended users, and to maintain it during usage.

a) Delivery (Stage)

The objective of this stage is to deploy the evaluated methodology to the development environment and conduct postmortem tasks. The activities are as follows:

Delivering: The produced methodology is delivered to its end users, ready to be enacted in software development projects. The necessary manuals and documents are produced, and training is conducted. The resources necessary for enacting the methodology (including tool support) are provided, and support and maintenance plans are produced.

Conducting postmortem: The lessons learnt from the project, including the problems encountered and their solutions, are documented for use in future SME projects.

b) Maintenance (Stage)

The purpose of this stage is to resolve the problems encountered during methodology enactment (*corrective* maintenance), to add new features to the methodology upon request (*perfective* maintenance), or to adapt the methodology to the changes made to the development environment and/or the situational factors (*adaptive* maintenance). Changes are applied to the methodology by executing the relevant stages of the Development phase.

IV. EXAMPLE

In this section, we demonstrate the enactment of parts of the UCDMD methodology through an example.

In the **Initiation** Phase, our example starts with identifying the situational factors and mapping them to requirements, as shown in Table I. Fig. 3 shows a use case diagram produced for this set of requirements.

TABLE I. EXAMPLE OF SITUATIONAL FACTORS AND REQUIREMENTS

Situational Factors	Degree of formalism required in the methodology
	Degree of developers' technical expertise
	Technology innovation level of the target system
Non-functional Requirements	Maintainability
	Risk management
Functional Requirements	Specify requirements
	Break down into tasks
	Design architecture
	Test
	Development

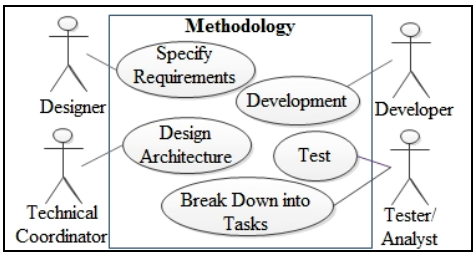


Figure 3. Example of a use case diagram for methodology development.

Use cases are then refined, and detailed descriptions are provided for each of them. Table II shows the particulars of the "Break Down into Tasks" use case. An architecture is then defined for the methodology; we have adopted the generic lifecycle [18] for our example. An important model produced in this phase is the interaction overview diagram, an example of which is shown in Fig. 4.

TABLE II. EXAMPLE OF A USE CASE DESCRIPTION

Use case: Break Down Into Tasks
ID: 3
Brief Description: The goal is to break down the requirements of the current iteration into fine-grained development (implementation) tasks.
Primary Actors: Analyst
Secondary Actors: None
Preconditions: - The requirements of the current iteration have been determined.
Main flow: 1. The use case is started when the Analyst requests the requirements of the current iteration to break them down into fine-grained tasks. 2. Methodology instructs Analyst on how to break down requirements. 3. For each requirement of this iteration: 3.1. Analyst breaks down requirement. 3.2. Methodology instructs Analyst on how to store the tasks. 3.3. Analyst stores the tasks. 3.4. Methodology instructs Analyst on how to evaluate the results. 3.5. Analyst evaluates the results.
Postconditions: Fine-grained tasks are ready for the current iteration.
Alternative Flows: - Suspend breaking down into tasks.

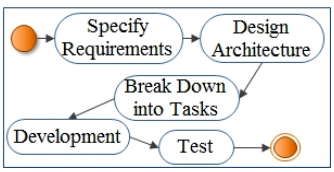


Figure 4. Example of an interaction overview diagram.

 ISBN: 978-1-61208-367-4

In the **Development** phase, type-independent analysis is first performed. Design models are produced after defining a type for the methodology: In our example, an agile methodology has been targeted; therefore, an agile process framework (from [22]) has replaced the initial architecture. The design class diagram of our example, and the design sequence diagram for "Break Down into Tasks", are shown in Fig. 5 and Fig. 6, respectively. The methodology is then implemented based on the design models (see Fig. 7).

V. EVALUATION

In order to gain a better understanding of the merits of the methodology proposed herein, we have conducted a criteria-based evaluation of UCDMD; the results are shown in Table III. The evaluation is based on the following evaluation criteria, specially designed to check the methodology for traits which a use-case-driven SME methodology would be expected to exhibit: Use-case-related [14], RE-related [2], general methodology-related [23], and SME-related [15]. It can be observed that UCDMD satisfies most of the criteria, faring especially well in the use-case-related, RE-related and SME-related categories.

VI. CONCLUSION AND FUTURE WORK

Using an object-oriented, use-case-driven approach for SME is a step forward; due to their functional nature, use cases can be mapped to the coarse-grained activities which form a methodology. On the other hand, using the object-oriented paradigm provides SME with the numerous benefits that the approach entails, including enhanced reusability, encapsulation, and flexibility. Moreover, our approach is also beneficial in facilitating the provision of tool support: The models produced can be directly used for implementing bespoke software support for the methodology.

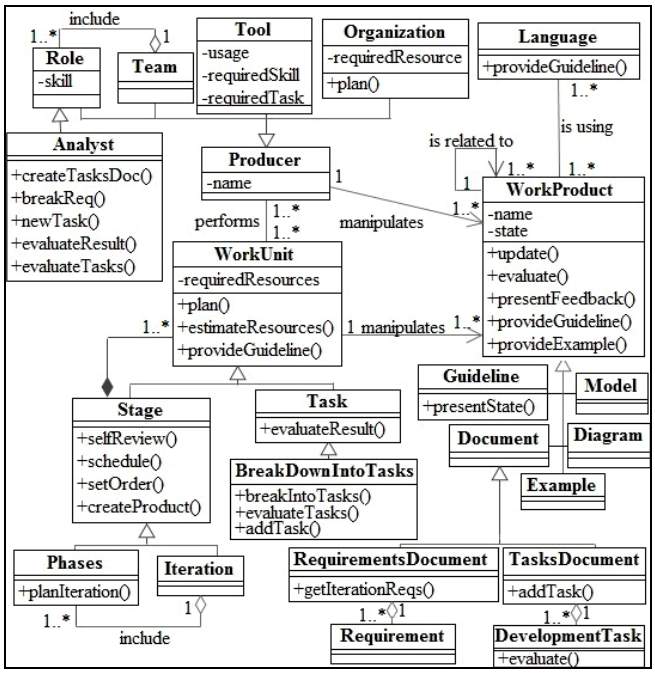


Figure 5. Example of a design class diagram.

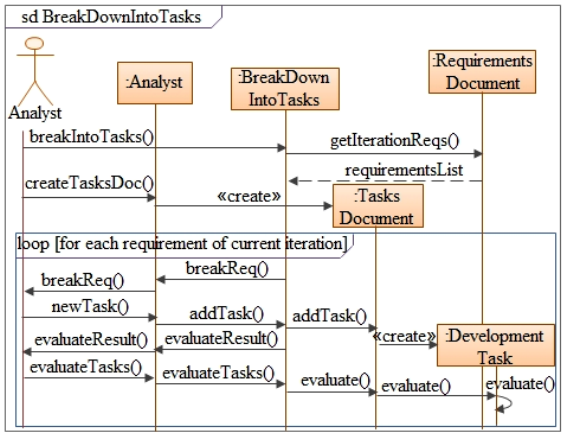


Figure 6. Example of a design sequence diagram.

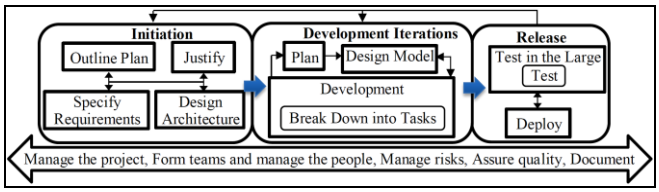


Figure 7. Example of an implemented methodology (lifecycle view).

Future work can be focused on applying UCDMD in an industrial-scale SME project. A parallel strand can proceed with refining and enhancing the tool production features of the approach. Future research can also focus on classifying the use cases typically encountered in SME projects.

ACKNOWLEDGMENT

We wish to thank Mr. Mohammad Reza Besharati for reviewing the Example and Evaluation sections.

REFERENCES

[1] J. Ralyté, S. Brinkkemper, and B. Henderson-Sellers, Situational Method Engineering: Fundamentals and Experiences. Springer, 2007.

[2] O. Jafarinezhad and R. Ramsin, "Development of Situational Requirements Engineering Processes: A Process Factory Approach," Proc. IEEE Computer Software and Applications Conf. (COMPSAC 12), 2012, pp. 279–288, doi: 10.1109/COMPSAC.2012.39.

[3] H. Gomaa, Software Modeling and Design: UML, use cases, patterns, and software architectures. Cambridge University Press, 2011.

[4] R. Ramsin and R.F. Paige, "Process-centered review of object-oriented software development methodologies," ACM Comput. Surv., vol. 40, Feb. 2008, pp. 1–89, doi: 10.1145/1322432.1322435.

[5] B. Westfechtel, Models and Tools for Managing Development Processes. Springer, 1999.

[6] H. Johnson, "An approach to software project management through requirements engineering," M.Sc. Thesis, Texas Tech University, 2010.

[7] C. Hug, A. Front, D. Rieu, and B. Henderson-Sellers, "A Method to Build Information Systems Engineering Process Metamodels," J. Syst. Softw., vol. 82, Nov. 2009, pp. 1730–1742, doi: 10.1016/j.jss.2009.05.020.

[8] M.D. Taufan, "Method Management System: Rule-Based Method Enactment Using MediaWiki and Semantic MediaWiki," M.Sc. Thesis, Radboud University Nijmegen, 2011.

[9] F. Karlsson and K. Wistrand, "Combining method engineering with activity theory: theoretical grounding of the method component

 ISBN: 978-1-61208-367-4

concept," Eur. J. Inf. Syst., vol. 15, Jan. 2006, pp. 82–90, doi: 10.1057/palgrave.ejis.3000596.

[10] J. Ralyté, "Requirements definition for the situational method engineering," Proc. Conf. Engineering Info Systems in the Internet Context, 2002, pp. 127–152, doi: 10.1007/978-0-387-35614-3_9.

[11] R. Ramsin and R.F. Paige, "Iterative criteria-based approach to engineering the requirements of software development methodologies," IET Software, vol. 4, Feb. 2010, pp. 91–104, doi: 10.1049/iet-sen.2009.0032.

[12] T. Olsson, J. Doerr, T. Koenig, and M. Ehresmann, "A Flexible and Pragmatic Requirements Engineering Framework for SME," Proc. International Workshop on Situational RE Processes, 2005, pp. 1–12.

[13] M. Asadi and R. Ramsin, "Patterns of Situational Method Engineering," Proc. Software Engineering Research, Management and Applications Conf. (SERA 09), 2009, pp. 277–291, doi: 10.1007/978-3-642-05441-9_24.

[14] A. Cockburn, Writing Effective Use Cases. Addison-Wesley, 2001.

[15] H. Agh and R. Ramsin, "Pattern-Based Model Transformation Method for Applying Model-Driven Development to Method Engineering," unpublished, 2014.

[16] B. Henderson-Sellers and J. Ralyté, "Situational Method Engineering: State-of-the-Art Review," Journal of Universal Computer Science, vol. 16, Feb. 2010, pp. 424–478, doi: 10.1.1.165.7993.

[17] Z. Shakeri Hossein Abad, M. Hasani Sadi, and R. Ramsin, "Towards tool support for situational engineering of agile methodologies," Proc. International Asia-Pacific Software Engineering Conf. (APSEC 10), 2010, pp. 326–335, doi: 10.1109/APSEC.2010.45.

[18] R.S. Pressman, Software Engineering: A Practitioner's Approach, 7th ed. McGraw-Hill, 2009.

[19] D. Firesmith and B. Henderson-Sellers, The OPEN Process Framework: An Introduction. Addison-Wesley, 2001.

[20] S.W. Ambler, Process patterns: Building large-scale systems using object technology. Cambridge University Press, 1998.

[21] E. Kouroshfar, H. Yaghoubi Shahir, and R. Ramsin, "Process patterns for component-based software development," Proc. International Symp. Component-Based Software Engineering (CBSE 09), 2009, pp. 54–68, doi: 10.1007/978-3-642-02414-6_4.

[22] S. Tasharofi and R. Ramsin, "Process patterns for agile methodologies," in Situational Method Engineering: Fundamentals and Experiences, J. Ralyté, S. Brinkkemper, and B. Henderson-Sellers, Eds. Springer, 2007, pp. 222–237, doi: 10.1007/978-0-387-73947-2_18.

[23] M. Taromirad and R. Ramsin, "CEFAM: Comprehensive Evaluation Framework for Agile Methodologies," Proc. IEEE Software Engineering Workshop (SEW 08), 2008, pp. 195–204, doi: 10.1109/SEW.2008.19.

TABLE III. RESULTS OF CRITERIA-BASED EVALUATION

	Criterion Name	*Criterion Definition*	*Possible Values*	*UCDMD Evaluation*
Use-Case-Related Evaluation Criteria	**Descriptive potential**	Is it possible to describe all functional requirements as use cases?	Yes/Partially/No	Yes
	Use case traceability	Are the work-products traceable to use cases?	Yes/No	Yes
	Flow modeling	Are use case steps modeled?	Yes (techniques), No	Yes (activity diagrams)
	Review and revision	Is the use case model reviewed/revised during the process?	Yes/No	Yes
	Mapping of actors to roles/teams	Can the actors be mapped to different roles/teams?	Yes/No	Yes
	Applicability	Are any specific patterns/guidelines provided for applying the use cases in SME?	Yes/No	Yes
Requirements-Engineering-Related Evaluation Criteria	**Requirements prioritization**	On what bases are the requirements prioritized?	Architectural value, Functional value, Business value, Development risk	Business value, Development risk
	Basis in requirements	Is the development process based on the requirements?	Yes (techniques), No	Yes (driven by use cases)
	Requirements change	Does the development process allow changes to the requirements?	Yes (techniques), No	Yes (use cases are updated at the start of each iteration)
	Realization of non-functional requirements	How are the non-functional requirements realized?	Mechanisms	Mapping to functional requirements, methodology type, or techniques
General Methodology-Related Evaluation Criteria	**Process definition**	Does the methodology explain the details of the development process?	Explicitly, Implicitly, No	Explicitly
	Quality assurance	Does the methodology support quality assurance activities?	Yes (techniques), No	Yes (traceability**,** continuous verification/validation, iterative process)
	Risk management	Does the methodology support risk management techniques?	Yes (techniques), No	Yes (continuous verification/ validation, iterative process)
	Flexibility	Does the methodology allow the process and modeling language to be tuned during its execution?	Yes (how), No	Yes (through reviews at the end of each iteration)
	Tool support	Is tool support provided or facilitated?	Yes (how), No	Yes (models facilitate the implementation of tools)
SME-Related Evaluation Criteria	**Traceability to situational factors**	Can the products be traced to situational factors?	Yes, No	Yes
	SME lifecycle coverage	Which phases of the generic lifecycle are covered by the development process?	*Analysis*, *Design*, *Implementation*, *Test*, *Deployment*, *Maintenance*	*Analysis*, *Design*, *Implementation*, *Test*, *Deployment*, *Maintenance*
	Support for SME strategies	Which SME approaches/strategies [16] are supported by the development process?	*Assembly-Based*, *Extension-Based*, *Paradigm-Based*, *Hybrid*, *Roadmap-Driven*	*Assembly-Based*, *Extension-Based*, *Paradigm-Based*

Insights from the Defect Detection Process of IT Experts: A Case Study on Data Flow Diagrams

Gul Tokdemir
Computer Engineering Department
Cankaya University
Ankara,Turkey
e-mail: gtokdemir@cankaya.edu.tr

Nergiz Ercil Cagiltay
Software Engineering Department
Atilim University
Ankara,Turkey
e-mail: necagiltay@gmail.com

Ozkan Kilic
Informatics Institute
Middle East Technical University
Ankara, Turkey
e-mail: ozkankilic@gmail.com

Abstract— **Design diagrams employed in software development process deliver groups of associated information about the software to be developed. They enhance the perception of the software engineers helping them better understand the software system at various levels of system development process. Today's fast-changing business environment necessitates the reflection of these changes into the operational software systems. Hence, the changes needed in software systems require software engineers to understand the system design diagrams and update them according to the changes. Therefore, it is very important for software engineers to understand and construct the design representations reflecting the software requirements correctly for the success of a software project. In the literature, there are not many studies conducted to better understand the behaviors of software engineers during designing and understanding these representations. Hence, the main aim of this study is to analyze the defect detection process of software engineers during their understanding of Data Flow Diagram (DFD) representations which are used to reveal system processes at different levels of abstraction and data flow requirements between them. Mainly, the question which type of defects can be detected easily is aimed to be answered. The results of this study show that missing information type defects (Missing Process-MP and Missing Dataflow-MD) are harder to detect than the incomplete or incorrect type (incorrect or missing Information-I) of defects.**

Keywords-DFD; software design; diagrammatic reasoning; defect detection.

I. INTRODUCTION

Diagrams can be more influential than sentential representations depending on the usage [1], as they communicate, and leverage knowledge that is crucial for solving problems [2]. Diagrams provide condensed information; hence, they are very effective in information systems for transferring information between stakeholders of the system during the system design phase. Moreover, during the software engineering lifecycle phases, they may offer reductions in cost and enhancements in understanding of the system.

During software development, engineers need to understand the system design from the diagrams, transform the system view into programs by viewing whole system, and check for consistency and errors resulting from misunderstanding of the design. As the understanding level of the engineers gets higher, their error correction performance is expected to increase. Finding and correcting these design errors or inconsistencies have a paramount effect in successful system development on time and within the predicted cost.

The aim of this study is to analyze the defect detection process by the software engineers during their DFD reviewing process. We believe that such analysis would provide insights about the design diagrams and software engineer's defect detection process. The results of this study are expected to provide insights to the researchers, software companies, and to the educators to improve DFD cognitive process. The State of the art section below contains related studies found in the literature, Methodology section explains the experiment, Result section analyzes the experiment results and Discussion and Conclusion section talks about the insights gained through this study.

II. STATE OF THE ART

Studies report that 40–50% of the development effort is being spent for fixing errors that could be detected and fixed early in the software development process [3]. Hence, defect detection performed early in the software development process is, an essential task as undiscovered defects may cause critical problems later in the process. In this regard, there are many studies mentioning defect detection activity as important, because, as they disseminate to the subsequent development phases, recovery would be more costly and difficult [4][5][6].

Studies also report that, by using model-based approaches, the defect detection rate could be increased in the early stages of the software life-cycle [7][8][9]. Accordingly, many researchers analyzed engineers'

 ISBN: 978-1-61208-367-4

perception of design diagrams and defect detection process of software engineers in ERD [10], DFD [10], and UML [4] and their cognitive processes [11]. For instance, Hungerford et al. [10] states that practice and proficiency in diagrams improve defect detection process of software engineers. Kumaresh and Baskaran [5] report that analysis of the defects at early stages of the software development lifecycle reduces development time, development cost and the resources required for the process.

Even though the DFD modeling language is over 30 years old, because of its usage history and familiarity among the software developers, many researchers, today, based their studies on this notation [12][13]. Additionally, since most of the current software systems' documentations are based on the DFD notations, for maintenance procedures the technicians are still required to better understand this notation. For instance Yuwen and Wang [14] report the drawing of DFD is the key technology in the development of system analysis and design [14]. According to them, DFD is not only the key composing part of the logic model in new system, but also the key basis in the system physical designing [14].

However, in the literature, there are not many studies conducted to better understand the reviewers' performance during the defect detection process. For instance, Moser and Biffl report that the missing or incorrect type of information is often detected in a later engineering process step [15]. Hence understanding the defect types that cannot be detected easily could help the software system designers to better represent this type of information in their representations. Additionally, this information also can be used to better guide the reviewers in different phases of software development process accordingly.

Hence, in this study, defect detection process of software engineers during their DFD reviewing process is analyzed to obtain insights about the cognitive processes of the engineers. Mainly, three different types of defects, namely, Missing Process (MP), Missing Dataflow or information (MD) and incorrect or missing Information (I) have been seeded into the DFD representations. The following research question is aimed to be answered is 'Which types of defects (MD, I, or MP) are easy to detect in DFD representations?'

Data are collected through interviews and observations while the IT experts work on the corresponding materials in defect detection.

III. METHODOLOGY

The experimental study is conducted with 4 participants using a study material which is derived from the study of Hungerford et al. [10], which is adapted to the current settings of this study and translated into Turkish. Participants of this study were software engineers with average age of 32 (Table 1).

TABLE I. PARTICIPANTS' INFORMATION

Participant	Age	Experience in the field	Gender	Experience with DFD
P1	29	8	F	8
P2	28	7	M	1
P3	34	12	M	2
P4	35	12	M	3
Average	32	10		3

We have prepared two DFDs of the system with 17 defects seeded in total at two levels. The participants have been provided the system description one week before the experiment. During the experiment, participants were asked to find the defects seeded in the DFDs, based on the system description.

The defects are categorized into three types: MP, MD and I. Table 2 summarizes the number of defects in the DFDs according to each category defined here.

TABLE II. NUMBER OF DEFECTS IN EACH CATEGORY

Code	Description	# of Defects
MP	Missing Process	2
MD	Missing Dataflow/information	9
I	Incorrect/ Incomplete	6
	Total	17

Table 3 depicts the defects seeded into both DFDs with their defect types. Figure 2 shows the locations of the defects (Fig. 1) at level 1 and 2 (Fig. 2).

TABLE III. DEFECT EXPLANATIONS

Defect	Description	DFD	Defect Type
01	End of job proposal process (1.4) is missing	1	MP
02	The entity named accounting should be job costing section	1	I
03	Job request data should go from customer to 1.1. Job Evaluation Process	1	MD
04	Receipt information should go from process "1.5 Payment Monitor" to the Customer entity	1	MD
05	Job proposal data flow should go data store named D2, instate of entity named accounting	1	I
06	The data flow from data store D1 to process 1.2 should be part information not customer information	1	I
07	From entity named customer, to the missing process named end of job proposal (1.4), rejection information should go	1	MD
08	the missing process named end of job proposal (1.4) to the data storage named D2, end of job proposal information should go	1	MD
09	From process 2.1 to the process 2.2 purchase order information should go	2	MD
10	From the entity supplier to the process 2.2, approval date and time information should go	2	MD
11	The Data storage named D7 should be supplier account, not customer account	2	I
12	From the process 2.2 to the storage D5, instate of customer information, part	2	I

 ISBN: 978-1-61208-367-4

	information should go		
13	The direction of the data flow (order form) from the entity supplier to the process 2.2 is incorrect. It should be from the process 2.2 to the entity supplier	2	I
14	Process 2.3 delivery is missing	2	MP
15	From the data storage D8 to the missing process 2.3, order form information should go	2	MD
16	From the missing process 2.3 to the process 2.2, delivered part information should go	2	MD
17	From the missing process 2.3 to the data storage D1, delivered part information should go	2	MD

In Figure 1, there are five processes describing top level relationships and data flow between processes. These five processes define the top level diagram of an ERP sales function module of a company. They include request evaluation, proposal preparation, work order preparation, work order close-up and payment follow-up processes. These processes connected to each other through data flows. Moreover, data is accumulated in data stores called customer account, work order/proposal and personnel.

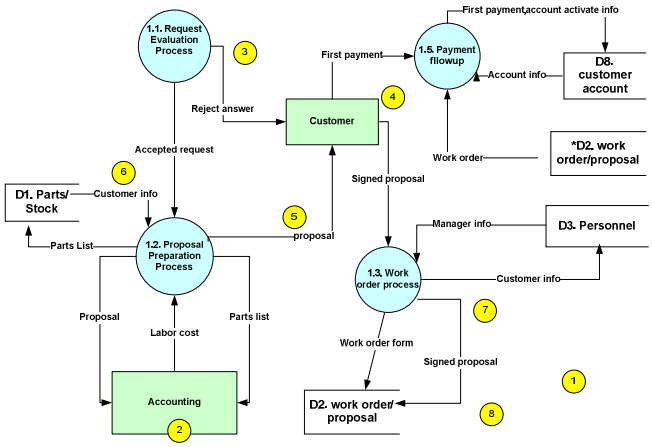


Figure 1. Defects' Placement in DFD_1

Similarly, Figure 2 depicts three sub-processes of proposal preparation process and their data flow. It has three processes which define second level DFD of proposal preparation process. They include parts/stock, order and delivery operations processes. These processes connected to each other through data flows. Moreover, data is accumulated in sx data stores called parts/stock, work order/ proposal, order form, customer account and supplier info.

As seen from Figures 1 and 2, the defects were seeded into two DFD diagrams and the participants were asked to detect them and take notes. During this process, the participants were allowed to check the system description. In the following section, the results of the defect detection process are provided.

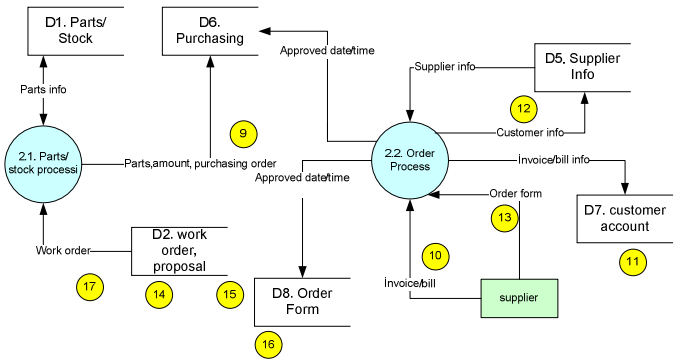


Figure 2. Defects' Placement in DFD_2

In this study, data is collected through Defect Detection Report used by the reviewers, observation notes and semi-structured interview sessions conducted by each reviewer. The defect detection report has the defect numbers and the explanation for the defects found. By using this form, the reviewers were asked to note each defect that they detect and describe their opinions about this defect as explained in the explanation document provided in Appendix A. The observations were conducted by one researcher and observation notes were taken during each reviewer's defect detection process. The durations spent for detecting each defect were recorded during the observation sessions and later synchronized with the reported defects in the Defect Detection Report. Additionally, by the same researcher, a semi-structured interview session was conducted by each reviewer individually. The interview sessions took around 30 minutes. The semi-structured interview questions were formed as below:

1. Which types of defects were easy to detect for you?
2. Which defects were hard to detect for you?
3. Which factors do you think helped you to detect the defects easily?
4. Which factors do you think maked it hard to detect the defects?

This study is conducted with the contribution of four participants who were asked to detect 17 defects seeded in two DFDs. Since the main research question of this study is based on the defects, the results of this study based on 68 cases (17 times 4). Additionally, this study aims to focus on the behaviors of the participants in order to uncover the complexity of human behavior in such a framework and present a holistic interpretation of what is happening during the review process. Nielsen and Landauer [16] also report that studying with four or five subjects is enough to understand and explain more than 80% of the phenomena. Accordingly, in this study, the participants' behaviors are analyzed in depth from different dimensions. Since each participant studied individually, we believe that this number of subjects could provide a view for understanding the phenomena.

 ISBN: 978-1-61208-367-4

IV. RESULTS

Table 4 shows the duration in seconds that each participant (D_{Pij}) spent during each defect detection process.

TABLE IV. DEFECT DETECTION DURATION DATA

Defect Type	Defect	Dp_{1j}	Dp_{2j}	Dp_{3j}	Dp_{4j}	AD_i
MP	14				993	993
I	13	386	60	678	70	299
MD	16				256	256
MD	07			145	236	191
I	12		114	347	69	177
MD	09		147			147
I	06	162	88	70	214	134
MP	01	133	113		114	120
MD	04	103				103
MD	03			89		89
I	02		6	163	47	72
I	11		45			45
I	05		36			36
MD	08					
MD	10					
MD	15					
MD	17					

As an example, in this table, D_{p1} is calculated from the observation data which shows the duration in seconds that the participant P1 spend time for detecting the defect i (D_i). It is the duration starting from the time point of last defect detection process until the defect detection of D_i. AD_i is the average of the durations spent by each participant to detect defect i (D_i). As seen in Table 4, the defects D_{08}, D_{10}, D_{15} and D_{17} were never detected. It is interesting that the defect type of all of these defects that were not recognized by any of the reviewers was MD type. On the other hand, most of the defects of type I, detected in relatively less time spent (D_2, D_5, D_{11}). Similarly, the participants spent more time for detecting defects D_{14} and only one participant could be able to detect this defect.

We have analyzed this data according to the defect types, as shown in Table 5. Accordingly, the detection rate for missing Information (I) type of defects is calculated as 16/24=0.67. Hence, defects of type I and MP were detected mostly; on the other hand the defects of type MD were detected seldom.

TABLE V. DETECTED DEFECT TYPE

Defect Type	Total Possibilities	Total Detected	Detection Rate
I	24	16	0.67
MP	8	4	0.50
MD	36	6	0.17

The detection frequency F_i of defects is shown in Table 6. In this table, F_i represents the frequency of a detected defect by participants. Its value is calculated by adding 1 point for each defect's detection for defect i (D_i). For example, if the defect is detected by only one participant this value is 1, if it is detected by three participants the Fi value for that defect is calculated as 3. As seen from Table 6, four defects 08, 10, 15 and 17 were never detected.

TABLE VI. DEFECT FREQUENCY F_I

Defect Type	Defect	F_i
I	06	4
I	13	4
MP	01	3
I	02	3
I	12	3
MD	07	2
MD	03	1
MD	04	1
I	05	1
MD	09	1
I	11	1
MP	14	1
MD	16	1
MD	08	
MD	10	
MD	15	
MD	17	

The average frequency of defect detection according to the defect types are given in Table 6. As seen from this table, the MD types of defects are detected less frequently, and the defect of type I detected most frequently. Parallel to this finding during the interviews, three reviewers (P2, P3, P4) reported that missing type of information were hard to detect. For instance, P3 reported that "the missing procedures were very hard to detect for me". Similarly, during the interviews, two reviewers (P1 and P2) reported that data flows were easy to understand. For instance P2 reported that "Detecting the data flow directions were easy. I easily detected the incoming and outgoing data. It was also easy to decide the data flow to each data store and which data should be read from a data store. Detecting the data, that supposed to go to a data-store but not shown in the design, was also easy". Moreover, we have asked participants about the factors that helped them to find the defects easily. They noted that the diagrams used to describe process were easy to detect. They stated that the data flows and external storages were difficult to follow in the diagrams. They said bigger and more detailed shapes with color would have increased the understandability of these diagrams.

V. DISCUSSION AND CONCLUSION

In this study, an experiment is conducted to analyze defect detection performance of software engineers in reviewing DFD diagrams. During the experiment, we had provided materials to the participants, one week before the experiment (Appendix A) and requested to find defects on DFD diagrams compared to the explanations given. They were asked to think aloud. We have recorded defect detection duration of each participant. The results of this study show that, missing information type defects (MP and MD) are harder to detect than the incomplete or incorrect type (I) of defects. Hence the defect detection frequency of defects in average is higher for of type I defects (2.67) that that of type MP (2.00) and type MD (1.20) defects. Similarly, the detection rate of type I defects (0.67) is higher than that of type MP (0.50) and type MD (0.70) defects.

 ISBN: 978-1-61208-367-4

According to the results of this study, the software system designers may reconsider their designs especially for the defects of type missing information, which are harder to be detected in the future and may increase the cost of software projects. We believe that further analysis of the DFD defect detection process is expected to provide more insights to the researchers, software companies, and to the educators to improve DFD cognitive process.

REFERENCES

[1] J.H. Larkin and H.A.Simon, "Why a diagram is (sometimes) worth ten thousand words," Cognitive Science, 1987, vol. 11, pp. 65-99.

[2] J. Zhang, "The nature of external representation in problem solving," Cognitive Science, 1997, vol. 21 i2. 179-217.

[3] B. Boehm and V.Basili, "Software defect reduction top 10 list", IEEE Computer, vol. 34, pp. 135–137, January. 2001.

[4] O. Laitenberger, C. Atkinson, M. Schlich, and K. El Emam, "An experimental comparison of reading techniques for defect detection in UML design documents," Journal of Systems and Software, August. 2000, vol. 53 n.2, pp. 183-204.

[5] S. Kumaresh and R. Baskaran, "Defect analysis and prevention for software process quality improvement," International Journal of Computer Applications, 2000, vol. 8 i7. 42L 47.

[6] G. Travassos, F. Shull, M. Fredericks, and V.R. Basili, "Detecting defects in object-oriented designs: using reading techniques to increase software quality," ACM SIGPLAN Notices, October. 1999, vol. 34 no. 10, pp. 47-56.

[7] R. Alur and A.Chandrashekharapuram, "Dispatch sequences for embedded control models", In Proc. 11th IEEE Real-Time and Embedded Technology and Applications Symp. 2005, vol. 11, pp. 508–518.

[8] L. Kof "Scenarios: identifying missing objects and actions by means of computational linguistics", In Proc 15th International Requirements Engineering Conference, pp. 121–130, 2007.

[9] L. Kof, R.Gacitua, and M. Rouncefield, P.Sawyer, "Ontology and model alignment as a means for requirements validation", in International Conference on Software Engineering, pp. 46–51, 2010.

[10] B.G.Hungerford, A.R.Hevner, and R.W.Collins,"Reviewing Software Diagrams: A Cognitive Study,", IEEE Transactions on Software Engineering, February, 2004, vol. 30 no. 2, pp. 82-96.

[11] K.A. Ericsson and H.A.Simon, Protocol Analysis: Verbal Reports as Data. revised edition, Bradford Books/MIT Press, Cambridge, MA 1993.

[12] F. Chan, "The Role and Mechanism of Analogical Transfers in Novices' Data Flow Diagram Problem Solving: The Effects of an Explicit Hint and Alternative Training Methods, Senior Honors Thesis, University of Hawaii, 2014.

[13] V. Repa, Object-Oriented Analysis with Data Flow Diagram. InInformation Systems Development (pp. 419-430), 2013, Springer, New York.

[14] S. Yuwen and K.Wang, A Method of Data Flow Diagram Drawing Based on Word Segmentation Technique. In Frontier and Future Development of Information Technology in Medicine and Education, pp. 3269-3274, 2014, Springer, Netherlands.

[15] T. Moser and S.Biffl, "Semantic tool interoperability for engineering manufacturing systems" In Proc. Emerging Technologies and Factory Automation (ETFA), IEEE Conference, pp. 1-8, 2010.

[16] J. Nielsen and T.K. Landauer, "A mathematical model of the finding of usability problems", Proc. ACM INTERCHI'93 Conference, pp. 206-213, 1993.

APPENDIX A

Problem Definition

Assume Mavi Company has business in pipe sector. The company's work and process descriptions are given below.

There are several types of employees working for Mavi Company, such as managers, sales staff and security guards. Telephones are shared and several employees may have the same office address. Security guards may be assigned to both buildings and car parks. Sales staff provides consultation services to customers by phone or face to face. Customers are assigned to exactly two salespersons and employees work with other employees in teams.

Each department can have more than one unit of the company. Personnel works in the units and each employee can work in one unit. Unit numbers and unit names are only defined uniquely in that department.

Customers can make job requests to Mavi Company. Mavi Company may reject this request, or if accepts, it prepares a job proposal and sends it to the customer.

When a job proposal is prepared, necessary parts' information is retrieved from parts file. Unit labor costs for parts are retrieved from job costing section. In this way, prepared job proposal is sent to the customer. Customer may accept or reject the proposal. If the customer rejects it, job proposal is closed. If accepted, the proposal is signed and the first payment is withdrawal.

Accepted job proposal is used to create a work order to follow the request in the company. For each customer's each job proposal, a single account is opened. A manager is appointed for each work order. Some work orders may include several customers. Orders associated with each other, brought together more than one job are classified as a new project. First invoice is sent to the customer at this step.

After the work orders are prepared, the necessary parts are controlled from the stock. If the parts do not exist in the stock, purchase is made using the amount information. According to the purchasing information, suppliers are identified; invoice is prepared and sent to the supplier. When the supplier approves the invoice, date and time is recorded. Each manufacturer must have a separate account. The supplier should provide invoice for the manufactured parts. This information is used to update the supplier info. Invoices are controlled as the parts are delivered. After the delivery, part information is updated in the stock.

Special promotional campaign is created for important projects. These campaigns are handled either by Mavi Company, or by a local organization like a school or an art festival. Projects cannot be performed by both campaigns. Each campaign introduces a single project.

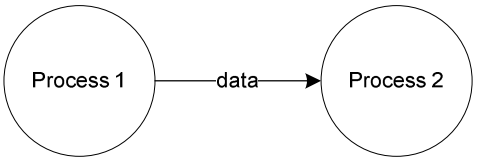
 ISBN: 978-1-61208-367-4

SOME DESCRIPTIONS ABOUT THE RESEARCH

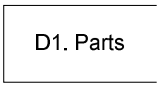
1. Assume you are employed to analyze the software system of Mavi Company. In this document, you are given information about the business process of Mavi Company.
2. You are required to use this information in analyzing the system to find the possible errors/mismatches. These errors/mismatches may exist because of incomplete or incorrect requirements.
3. The errors/mismatches you found should be based on the system definition and the other supporting documents presented to you earlier. Assume the document describes the company processes correctly.
4. In this study, you are not required to create new solutions to solve the problems or not required to fix these problems.
5. You are given 2 hours to find the induced errors/mismatches. Please adjust your time accordingly.
6. Identify errors/mismatches and list them on the forms provided. To describe the error/mismatch, if possible, please specify the related process(es) and data-flow information. If not possible, please use most appropriate way to explain the error/mismatch.
7. You can use any method or technique to find the Identify errors/mismatches. However, during the process, please don't interact with anyone else.
8. In identifying the errors/mismatches, you can review the documents provided to you as you want.
9. Please, try to think loudly as you are analyzing the system design. While you are reading and interpreting the documents, try to talk loudly. please, please. In particular, when you identify errors/mismatches, please indicate your findings loudly.

DFD Notations

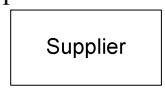
1. The DFD diagrams used in this study are developed by Visio. The processes are represented by circles; the data flow is represented through arrows as described below.



2. Data storage is represented as below:



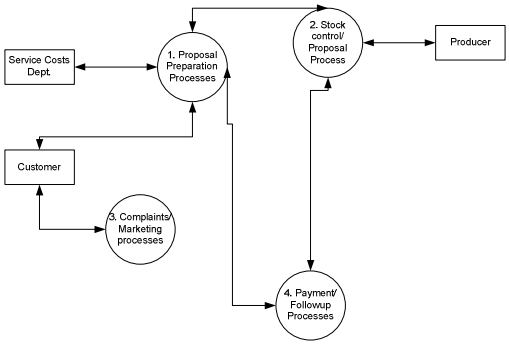
3. External entity is represented as below:



4. In this study, you are given Context diagram (Level-0 DFD) and DFD of two processes in detail (Proposal Preparation process and Stock control/ Proposal process).

5. There are 17 defects in DFD diagrams These can be missing process, Missing Dataflow/information, Incorrect/ Incomplete data flow type defects

6. The top level process definitions are given in the figure below. Proposal Preparation and Stock control/ Proposal processes' DFD will be given during the experiment.



 ISBN: 978-1-61208-367-4

Using Expert Systems for Coaching and Mentoring ICT Project Managers

Robert T. Hans
Department of Software Engineering
Tshwane University of Technology
Pretoria, South Africa
hansr@tut.ac.za

Ernest Mnkandla
School of Computing
University of South Africa
Florida, South Africa
mnkane@unisa.ac.za

***Abstract*— Several risks, dynamics and challenges, including lack of skilled and experienced personnel, mobility associated with project management experts and tough economic conditions are just some of many issues that information and communications technology (ICT) organizations in the 21st century have to deal with. Furthermore, these organizations are under constant pressure to improve project success rate which are unacceptably very low. Project managers who are ineffective in project leadership due to poor project skills pose a serious risk to project success. Some research studies indicate that the demand for ICT project managers with proper knowledge and expertise is ever increasing and the supply is nowhere near close in meeting the demand. As part of providing a solution to the abovementioned challenges and help equip ICT project managers with correct skills through mentoring and coaching, this research study is proposing the use of expert systems (ES). This proposal is a response to calls that have been made by other studies in project management that new approaches of developing project managers must be pursued. The use of expert systems to equip ICT project managers with the right skills and expertise will help advance and improve their software project management expertise. Just like 'intelligent organizations' that use expert systems to improve their decision-making processes in order to advance business efficiency and competitiveness so should expert systems be used to coach and mentor less experienced project managers. This research paper argues that the use of ES for coaching and mentoring yield many benefits for organizations.**

Keywords- expert system; intelligence system; project managers; coaching; mentoring; skills.

I. INTRODUCTION

Several risks, dynamics and challenges, such as, lack of skilled and experienced personnel, volatility of human experts [1] and tough economic conditions create a challenge for many information and communications technology (ICT) organizations in the 21st century. These ICT organizations are under constant pressure to improve project success rate through, amongst other things, the use of experienced project managers who have great influence on project success [2] to run their projects. These are some of the challenges that organizations in the ICT sector have to contend with in their quest to deliver value to both stakeholders and shareholders.

According to Schwalbe [3] the project management framework consists of nine knowledge areas, which describe the key competencies that project managers should possess in all the nine knowledge areas in order to deliver on projects' mandates. However, a study by Hans et al. [4] shows that ICT project managers in South Africa lack some key project management competencies, and these include problem-solving and leadership expertise. Project managers who are ineffective in project leadership due to poor project skills pose a serious risk to project success [5][6]. Project managers may lack appropriate expertise due to a number of reasons. They may lack skills because of not being properly trained or mentored and were just appointed based on their previous excellent performance in their former positions.

The above-mentioned challenges make it necessary for ICT organizations to rethink their business practices of training and mentoring their project managers. Moreover, talent development for project and program managers remains a top concern in organizations. This comes as no surprise given that research studies indicate that experience and project management expertise are key in delivering successful projects [3][7][8]. Metaxiotis [9] indicates that the demand for ICT project managers with proper knowledge and expertise is ever increasing and its supply is nowhere near close in meeting the demand. As part of providing a solution in equipping ICT project managers with appropriate skills through mentoring and coaching, this research study is proposing the use of expert systems. The proposed solution will result in project managers improving their software project management expertise. The use of the proposed ES would further address the skewed supply-demand ratio of knowledge-based resources – the supply of skilled project managers will be improved. It will also enable the training of project managers in the real-world project environment as requested by Ramazani et al. [10]. The use of the proposed ES will also relieve project management experts from the duties of mentoring and coaching which they sometimes do reluctantly [11]. Ramazani et al. [10] are calling for fresh approaches in the development of project managers, and this paper's proposal intends to fill that gap by proposing a new approach in training, coaching, mentoring and development of project managers through the use of expert systems. Even though expert systems have been used in other areas of the project management discipline, to the best knowledge of the authors of this paper expert systems have not been used for the development of project managers. Therefore, this research paper proposes a novel approach to use expert systems to provide the above mentioned services and thus address the challenges which are plaguing the project management discipline.

The remainder of this research paper proceeds as follows. Section II presents a research methodology used in this

 ISBN: 978-1-61208-367-4

study. Section III discusses expert systems and their application. Section IV presents a discussion on using expert systems for coaching and mentoring ICT project managers. Sections V presents the architectural structure of the proposed ES, while Section VI discusses value add which is derived from using expert systems for such initiatives. Finally, conclusions, limitations of this research study, and directions for future studies are presented.

II. RESEARCH METHODOLOGY USED

As pointed out previously, ICT project managers in South Africa lack some key project management competencies. This is a research problem which this study seeks to address. A research question which will assist in finding a suitable solution to the abovementioned problem is the following:

Can expert systems be used for equipping ICT project managers with appropriate skills?

An approach used by this research study to answer the abovementioned research question is through the exploration of relevant literature with the aim of establishing similar cases where expert systems have been successfully used. There is therefore neither data collection nor data analysis performed in this research study.

III. EXPERT SYSTEMS: THEORY AND PRACTICE

Organizations are continuously searching for innovative methods of reducing costs, improving decision making processes and automating or simplifying routine tasks. Therefore, organizational survival depends on finding ways and practices of adapting to the continuous changing competitive environment. Expert systems (ES) or intelligent systems are one type of IT tools that organizations turned to for addressing such challenges [12]. Avram [13] defines expert systems as:

"Systems that use knowledge-based techniques to support human decision-making, learning and action."

ES contains knowledge and experience of experts in a specific domain that anyone can use in solving problems [14]. Expert systems are a branch or subset of artificial intelligence [15][16]. Intelligent systems are considered 'intelligent' because they can solve a problem in a way similar to a human expert [17].

Expert systems have found application in a wide range of fields, such as manufacturing, business, finance, airline, law, computer science, geology, education, mathematics and medicine [16][18][19][20]. With each field, expert systems have been used to solve different range of problems. For example, some companies have implemented expert systems to assist in performance appraisal processes [21]. Others have used artificial intelligent systems for tutoring undergraduate auditing and engineering students at various universities [22]. Jenicke [17] cites three business organizations, namely, Digital Equipment Corporation, General Electric and Coopers & Lybrand that have developed and are using expert systems in their respective business domains. Digital Equipment Corporation uses its expert system called XCON [23] for configuring VAX systems which handle customer orders. XCON has resulted in an improved customer order processing for the company. General Electric uses an expect system called DELTA for diagnosing and repairing the company's diesel-electric vehicles which are used for railroad maintenance [17]. On the other hand, Coopers & Lybrand uses its expert system called ExpertTAX [17] for providing expert advice to the organization's accountants so that they may in turn respond intelligently to clients' tax related questions. The system acts as an intelligent advisor to the company's employees who seek its guidance for decision making. ExpertTAX stores the expertise of the company's experienced accountants in its knowledge base component. Metaxiotis [9] also makes mention of organizations, such as Singular and the Portuguese Railways that have successfully implemented expert systems in their business operations. The discussion above illustrates that an ES may be used to position an organization in a better strategic position in the marketplace.

The literature has many research studies which cite the usefulness of expert systems in assisting in decision making. Expert systems may also play an important role in retaining competitive project management knowledge which may be transferred to less experienced project managers. This assertion is also supported by Jenicke [17] who states that expert systems are suitable for dissemination of knowledge and expertise within their areas of application. The use of intelligent systems for coaching and mentoring purposes has become imperative because the supply of experienced project managers with correct expertise is in short supply [24] and some project managers lack some key project management competencies. The use of expert systems for equipping ICT project managers with appropriate skills would help improve the supply of project managers with the right skills set. Moreover, this proposal is also in line with what other previous studies have established in terms of using computer-based training for enriching employees, as well as learners with much needed knowledge [5][22].

The proposed approach in developing ICT project managers is in accordance with the Guide to the Project Management Body of Knowledge [25] framework for managing project human resources. The PMBOK Guide (2012:105) states:

"Individual development (managerial and technical) is the foundation necessary to develop the team."

Such development includes enhancing skills, knowledge and capabilities of team members through, for example, computer-based training [25]. Furthermore, according to the PMBOK Guide (2012) one of the primary functions of a project management office (PMO) is to ensure coaching, training and mentoring of project managers. This research paper is therefore in direct support of PMO functions.

Notwithstanding the differences in the role that a project manager plays under PRINCE2 and PMBOK [26] a project manager is still a key project stakeholder whose primary responsibility is to steer a project to success. Such an expectation therefore requires that a project manager be properly skilled and this is the aim of this research study.

 ISBN: 978-1-61208-367-4

IV. USING EXPERT SYSTEMS TO COACH AND MENTOR ICT PROJECT MANAGERS

The failure rate of projects is still unacceptably high and several studies have highlighted these cases of failed projects. On the other hand, Schwalbe [3] indicates that 97% of successful projects were carried out by experienced project managers who have correct project management skills. However, as mentioned above the sad part is that such project managers are in short supply [24] and therefore, these findings further back the call made by this paper for the use of expert systems to improve the situation.

It was mentioned above that expert systems have been used for tutoring students and therefore, using them for mentoring and coaching project managers would not be a wrong concept. Expert systems may either use rule-based (theory-based) or case-based (experienced-based) form of reasoning in solving problems [22]. Expert systems which make use of both rule-based and case-based techniques would prove useful in training ICT project managers. This assertion is based on the fact that people solve problems by either using prior cases or rules depending on the task being solved [22].

The use of ES for coaching and mentoring initiatives would be beneficial to an organization and to the mentored individuals in many ways. Firstly, it would ensure that valuable expert knowledge is kept and preserved in the knowledge base of an expert system for future use. Such practical real-world knowledge from experts provides aspiring ICT project managers with valuable learning experiences [27]. In the case of project management, such knowledge would pertain to project areas, such as [3][28]:

- knowledge in performing trade-off amongst project triple constraint [3],
- knowledge in project risk management,
- knowledge in scope management, as well as
- knowledge in other areas that pertain to the eleven knowledge areas of project management [28].

Furthermore, the knowledge base of the ES would also include expert knowledge in project management areas in which South African project managers were found to be lacking key expertise as identified by Hans et al. [4].

The lack of project management expertise by project managers in the abovementioned areas [5] have contributed to the failure of ICT projects [29]. For, instance, a study by Standish Group [30] indicates that less than a third of projects finish on time and within budget. This indicates that project managers have problems in dealing with two of the three project constraints. Another study by Ibbs et al. [28] shows that organizations in the ICT industry are struggling in managing project risks. Although this has been a well-known problem, little has been done by organizations to address it [3]. Therefore, the use of an expert system for mentoring and coaching project managers in this knowledge area will go a long way in addressing this issue. Project scope management is another key area that needs project managers to pay attention to. Schwalbe [3] indicates that proper project scope management is a contributing factor to project success. The discussion above justifies the inclusion of project knowledge on the stipulated areas above in the knowledge base of the ES expert system.

Secondly, once the desired knowledge has been kept in the ES knowledge base it can then be used to train, coach and mentor both inexperienced and aspiring project managers that an organization wants to groom. When an inexperienced project manager uses an expert system, he/she is able to learn (infer) how the system arrived at a particular correct decision. Through such interactions with an ES, a project manager is able to gain valuable real-world knowledge and experience. Furthermore, such knowledge transfer forms part of organizational culture transfer to 'new' project managers. The initiative of using ES to train, coach and mentor novice project managers would be playing a critical and imperative role of grooming new project managers in-house and thus ensuring and perpetuating consistency in the way an organization manages its projects. Previous research studies show that organizations which groom project managers internally are amongst those that run successful projects [3]. Therefore, expert systems would be playing a role of transferring knowledge and the problem-solving strategies of experts to less experienced project managers [22].

V. ARCHITECTURE OF THE PROPOSED EXPERT SYSTEM

According to Metaxiotis [9] and Jenicke [17], expert systems have the following three basic main components:

- **Knowledge base** – The knowledge base contains the knowledge needed for solving a specific problem. In order for ES to solve human problems, human expert knowledge should be captured in a knowledge base [16]. In this case an ICT expert project manager's knowledge will be captured into the knowledge base of the proposed ES. The previous section mentioned some project management knowledge that should form part of this component. The knowledge base may be in various forms, such as, facts, theories, heuristics or relationships. Typically, the knowledge base is implemented in IF-THEN rules [16]. The development of this component of an expert system is the most challenging one [14].
- **Inference engine** – This component takes the input that has been entered by the user through user interface and then manipulates knowledge base using the inference control procedure [17]. The control procedure determines the order in which the knowledge base will be searched [9]. It may start with a set of conditions and then establish a conclusion or it may start with a conclusion and then search the knowledge base for conditions that meet the conclusion [9][16][17].
- **User interface** – Through this component a novice project manager is able to interact with the expert system. He/she is able to ask the ES to test some conclusions or enter information which the system will use to find conclusions related to the entered

 ISBN: 978-1-61208-367-4

information [16]. In return, the expert system is able to present its results and possibly prompt the user for additional information via the same user interface. An expert project manager also uses this user interface to capture project management knowledge into the knowledge base.

Figure 1 depicts the components of the architecture of the proposed expert system, as discussed above.

VI. THE VALUE OF USING ES TO COACH AND MENTOR ICT PROJECT MANAGERS

Information technology has become an indispensable factor for every organization [31]. ES as an integral part of IT provides an organization with an excellent opportunity of managing knowledge in project management and also of enabling knowledge transfer to inexperienced project managers. Metaxiotis [9] stresses the importance of using information technology by organizations in order to gain a competitive advantage. Therefore, an organization can obtain a competitive advantage from the use of ES through knowledge retention (knowledge captured in ES) and knowledge transfer (through using ES to coach and mentor inexperienced project managers). A number of authors have noted some benefits of using expert systems for training human resources, and such benefits include:

- **Continuous availability of services** – services provided by expert systems are always available anywhere, unlike when such services are offered by a human expert which might not be available or their availability may be confined to a specific location [16].
- **Costs savings**. The maintenance of human expert knowledge through the conventional way (for example, training a new project manager) may be more expensive. Therefore, the use of ES for training, mentoring and coaching would result in costs savings for organizations, a view also supported by both [17] and [24]. The usage of ES reduces the training and knowledge transfer cycle amongst staff members [9].
- **Consistency** - expert systems apply reasoning consistently without any biasness unlike human beings. This would ensure that consistency is 'transferred' to mentored managers. That is, ES teaches project managers consistency in their reasoning process [9].
- **Keeping lessons learned and updating knowledge base** – with case-based expert systems current decisions and incidents may be stored for statistical and future case-based reasoning [14].
- **Actively creating intellectual capital** – the use of ES creates organizational knowledge which will enable the business to compete effectively. It facilitates the continuous training of new project managers for an organization. This contributes to the continuous availability of human resource reserves [32].
- **Promoting a learning culture in an organization and empowering project managers** – by adopting ES an organization does not only remain in the cycle of knowledge creation and knowledge sharing [17][32] but also becomes a learning organization and at the same time empowering its project managers.

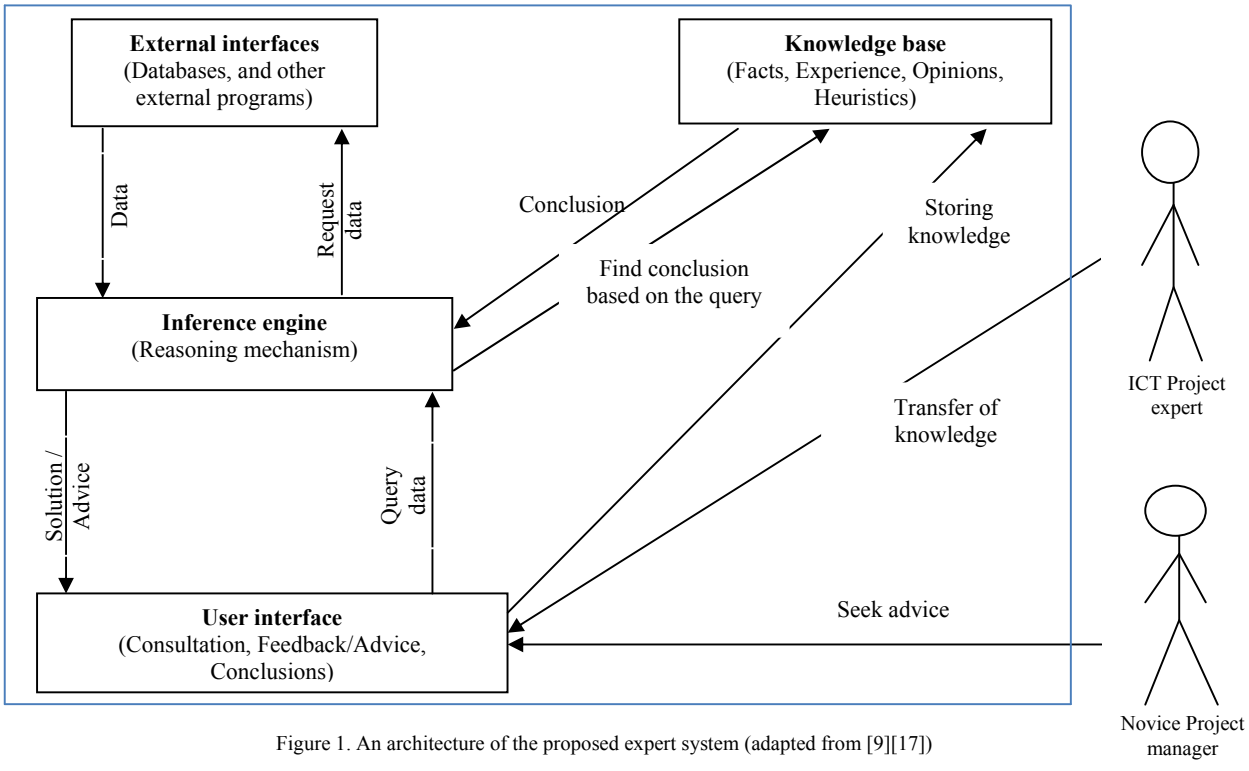


Figure 1. An architecture of the proposed expert system (adapted from [9][17])

- **Enabling on the job training (OJT)** – On the job training is very important for producing highly competent employees [33] and would be employees. In a competitive business environment where organization cannot afford to send away employees for training or they may have no capacity to provide such training [34], the usage of expert systems for training purposes enables such organizations to kill two birds with one stone – allows companies to be productive and also enables them to equip employees with much needed skills.
- **Provide learning anywhere and anytime –** ES enable any employee to learn anytime, and anywhere [35].

The use of expert systems further ensures efficient transfer of knowledge without costing an organization too much money and time. It also offers trainees a hands-on experience rather than being passive learners. This kind of training is consistent and repeatable, and these are the elements which are missing with training offered by human agents.

Over and above of the benefits mentioned on the usage of expert systems, herewith below are some of the specific benefits that accrue from the implementation of the proposed expert system for mentoring and coaching ICT project managers:

- **Improvement of project management efficiency**. Given the short supply of ICT project managers with appropriate and much needed project management skills [9][34], the proposed ES will not only ensure continuous supply of such managers but will also contribute to the better management of ICT projects. The proposed ES will focus on equipping project managers with the identified lacking skills which include problem-solving expertise, critical thinking, leadership, tools expertise, etc. [4][10]. In other words, the proposed ES will be aimed at mentoring and coaching ICT project managers on the identified critical skills gaps in project management.
- **Meeting both the 'demand and supply sides' of knowledgeable ICT project managers -** It was indicated earlier in this paper that currently there is a skewed supply-demand ratio of knowledgeable ICT project managers. The use of the proposed ES will seek to balance this unfavorable supply-demand ratio.
- **Facilitation of sharing of specific project management know-how –** Captured ICT expert know-how will be transferred through the proposed ES to new or novice project managers. As indicated before in this paper such project management knowledge is critical to project success and project managers with such knowledge are in short supply and thus the use of the proposed ES for addressing this will be a welcome relief.
- **Equipping project managers in the real-world context –** Project managers will learn project management skills in the real world, as the expert system will be based on real life experiences of project management experts. A study by Ramazani et al. [10] highlights the need to train project managers in the environment and context they are likely to encounter at work.
- **Assists in overcoming some of the barriers encountered by conventional knowledge sharing –** There are various barriers to knowledge sharing, including lack of socialization among staff members within organizations. Such a barrier inhibits transfer of tacit knowledge between staff members, however the use of the proposed ES would overcome such barriers as knowledge would be residing in the ES system.

Furthermore, the use of the proposed ES to part knowledge to new project managers lessens the approach of learning through trial and error that project managers are sometimes subjected to.

VII. CONCLUSION AND FUTURE WORK

If organizations hope to address the plight of ICT projects' poor track record and the lack of skilled project managers, then they need to change their development strategies of project managers. These sentiments have been echoed by [10] who state that new approaches are needed in developing project managers. According to Kilkelly [24],

"Project management is complex, and so to create good project managers and, subsequently, sound projects, it is critical to get the development right".

The use of the proposed expert system for training, mentoring and coaching new inexperienced ICT project managers is one way of getting the development strategy right as well as proposing a new way of training project managers as called by Ramazani et al. [10]. The usage of an expert system enables organizations to promote and maintain excellence through knowledge management, coaching and mentoring of their future project managers with the aim of creating a better future for the stakeholders.

The discussion presented above has attempted to answer the question of whether expert systems can be used to equip ICT project managers with much needed skills in project management. This research paper has argued that the use of expert systems for coaching and mentoring new and novice ICT project managers yields many benefits for ICT organizations, with the main benefit being the enablement of the transference of much needed project management know-how between expert and novice project managers.

This research paper is part of the PhD research work of the first author, where he intends to develop real-time interactive project management intelligence (PM*Int)* tool which is modelled after business intelligent tools [36]. Once the PM*Int* tool has been developed, it will then be tested for its effectiveness in improving decision making.

The usefulness of the proposed expert system for coaching and mentoring novice information and communications technology project managers needs to be tested. The first challenge though is getting participation of ICT project management experts when the proposed expert system is developed. The experts might view the system as meant to replace them [37] and this may lead to their lack of participation in the development of the system. Secondly,

 ISBN: 978-1-61208-367-4

the complexity and exorbitant costs associated with the development of expert systems [11] might be a challenge for some organizations.

REFERENCES

[1] S. Kutti, B. Garner, and A. Ghosal, "Modelling expert resource management systems", Kybernetes, vol. 28, issue 4, 1999, pp. 385-406.

[2] C. Standing, A. Guilfoyle, C. Lin, and P. E. D. Love, "The attribution of success and failure in IT projects", Industrial Management & Data Systems, vol. 1061, issue 8, 2006, pp. 1148 – 1165.

[3] K. Schwalbe, Information Technology Project Management, 6th ed., USA: Thomson Course Technology, 2011.

[4] R.T. Hans and P.M.D. Rwelamila, "Knowledge Base of Project Managers in the South African ICT Sector", Computer Science and Information Technology, vol 2, vol. 1, January, 2012, pp. 455-478.

[5] L.A. Kappelman, R. Mckeeman, and L Zhang, "Early warning signs of IT project failure: The dominant dozen", EDPACS, ABI/INOFRM Global, vol. 35, issue 1, 2007, pp. 1-10.

[6] L. Tichy, and T Bascom, "The Business End of IT project failure", Mortgage Banking, ABI/INOFRM Global, vol. 68, issue 6, 2008, pp. 28-35.

[7] B. Little, "The principles of successful project management", Human Resource Management International Digest, vol. 19, issue 7, 2011, pp. 36-39.

[8] J. Johnson, K.D. Boucher, K. Connors, and J. Robinson, "Collaboration: Development & Management", Software Magazine – February/March, pp. 1-9, 2001.

[9] K. Metaxiotis, "Leveraging expert systems technology to improve service industry", European Business Review, vol. 17, issue 3, 2005, pp. 232-241.

[10] J. Ramazani and G. Jergeas, "Project managers and the journey from good to great: The benefits of investing in project management training and education", International Journal of Project Management, http://dx.doi.org/10.1016/j.ijproman.2014.03.012, article in press.

[11] K. Annaiahshetty and N. Prasad, "Expert System for Multiple Domain Experts Knowledge Acquisition in Software Design and Development", 15th International Conference on Computer Modelling and Simulation, 2013, pp. 196-201.

[12] T.A. Byrd, F. Shieh, and T.E. Marshall, "The development and implications of the COMMU expert system", Industrial Management & Data Systems, vol. 96, issue 3, 1996, pp. 11-16.

[13] G. Avram, "Empirical Study on Knowledge Based Systems", Electronic Journal of Information Systems Evaluation, vol. 8, issue 1, 2005, pp. 11-20.

[14] M.A. Mach and A. M. Salem, "Intelligence Techniques for business intelligence in healthcare", IEEE Computer Society, 10th International Conference Intelligent Systems Design and Applications, 2010, pp. 545-550.

[15] B.L. Raggad and M.L. Gargano, "Expert System: defection and perfection", Logistics Information Management, vol. 12, issue 5, 1999, pp. 395 – 406.

[16] I.M. Shaluf and F. Ahamadun, "Technological emergencies expert systems (TEES)". Disaster Prevention and Management, vol. 15, issue 3, 2006, pp. 414 -424.

[17] L.O. Jenicke, "The Expert Systems as a Decision Support Tool", American Journal of Business, vol. 3, issue 1, 1998, pp. 47-52.

[18] E.T. Lee, "Intelligent factories using fuzzy expert systems". Kybernetes MCB University Press. vol. 25, issue 3, 1996, pp. 51-55.

[19] D. Corney, "Food bytes: intelligent systems in the food industry", British Food Journal, vol. 104, issue 10, 2002, pp. 787-805.

[20] H. Wu, Y. Liu, Y. Ding, and Y. Qiu, "Fault diagnosis expert system for modern commercial aircraft", Aircraft Engineering and Aerospace Technology, vol. 76, issue 4, 2004, pp. 398-403.

[21] M.G. Martinsons, "Human Resource Management Applications Knowledge-based Systems", International Journal of Information Management, vol. 17, issue 1, 1997, pp. 35-53.

[22] N. Wongpinunwatana, C. Ferguson, and P. Bowen, "An experimental investigation of the effects of artificial intelligence systems on the training of novice auditors" Managerial Auditing Journal, vol. 15, issue 6, 2000, pp. 306-318.

[23] J. Liebowitz, "Knowledge-based/expert systems technology in life support systems", Kybernetes, vol. 26, issue 5, 1997, pp. 555 – 573.

[24] E. Kilkelly, "Blended learning: pathways to effective project management", Development and learning in organizations, vol. 23, issue 1, 2009, pp. 19-21.

[25] Project Management Institute, A Guide to the Project Management Body of Knowledge (PMBOK Guide), 5th ed., 2012.

[26] R.M. Wideman, "Comparing PRINCE2 with PMBoK®", AEW Services, 2002, pp. 1-11.

[27] J.B. Arbaug, "Introduction: Project Management Education: Emerging Tools, Techniques, and Topics", Academy of Management Learning & Education, vol. 6, issue 4, 2007, pp. 568-569.

[28] C. W. Ibbs and Y. H. Kwak, "Assessing Project Management Maturity", Project Management Journal, vol. 31, issue 1, 2000, pp. 32 - 43.

[29] L. Tichy, and T. Bascom, "The Business End of IT Project Failure", Mortgage Banking, pp. 28, 2008.

[30] Standish Group, "Chaos Report", West Yarmouth, MA: The Standish Group International Inc, 2004.

[31] S. Indrajani and Y. Lisanti, "Business Intelligence Design on The Company", IEEE Computer Society, Fourth International Conference on e-Education, Entertainment and e-Management, 2011, pp. 307-310.

[32] D.P. Nawinna, "Fostering Business Intelligence in Banks through Knowledge Management: A case of Sri Lanka", Proceedings of the International Conference on Business Management & Information Systems, 2012, pp. 377-385.

[33] T. Nakamura, Y. Kitaura, H. Maruyama, and A. Takashima, "Analysis of Learners' Behavior in Role-play Training for Project Management Education", Ninth IEEE International Conference on Advanced Learning Technologies, 2009, pp. 144-146.

[34] Y. Tachikawa, H. Maruyama, T. Nakamura, and A. Takashima, "A Method for Evaluating Project Management Competency Acquired from Role-Play Training", IEEE Global Engineering Education Conference, 2013, pp. 162-170.

[35] Z. Chunua, "E-Learning: The New Approach for Knowledge Management (KM)", International Conference on Computer Science and Software Engineering, 2008, pp. 291-294.

[36] R.T. Hans and E. Mnkandla, "Modeling Software Engineering Projects as a Business", IEEE AFRICON 2013 Conference, 2013, pp. 1172-1176.

[37] T.A. Byrd, "Expert systems implementation: interviews with knowledge engineers", Industrial Management & Data Systems, vol. 95, issue 10, 1995, pp. 3-7.

 ISBN: 978-1-61208-367-4

Towards Autonomic Context-Aware Computing for SaaS Through Variability Management Mechanisms

Asmae Benali, Bouchra El Asri and Houda Kriouile

IMS Team, SIME Laboratory
ENSIAS, Mohammed V University
Rabat, Morocco
{asmae.benali, houda.kriouile}@um5s.net.ma
elasri@ensias.ma

***Abstract*—Owing to the multi-tenancy of Software-as-a-Service applications, the management of their resources becomes a challenge and a crucial task in order to provide highly configurable applications to thousands of tenants in a shared and heterogeneous cloud environment. They need dynamic context-aware configuration and intelligent strategies for provisioning available and cost-efficient services. In this sense, this paper identifies open issues in autonomic resource provisioning and shows innovative management techniques for these applications on cloud. Indeed, our work will focus on implementing an autonomic management artifact of services variability concerning the context. In this paper, we highlight our process for the development of autonomic context-aware to manage the SaaS variability.**

Keywords--multi-tenancy; context-aware; autonomic system; SPL; SaaS

I. INTRODUCTION

The emergence of SaaS (Software-as-a-Service) provision and cloud computing in general had recently a tremendous impact on corporate information technology.

While the implementation and successful operation of powerful information systems continues to be a corner stone of success in modern enterprises, the ability to acquire IT (Information Technology) infrastructure, software, or platforms on a pay-as-you-go basis has opened a new avenue for optimizing operational costs and processes. Cloud computing as defined by the NIST [1] as an IT model that allows network to have an easy access to a shared set of configurable computing resources. Cloud Computing providers offer their services in three basic models: SaaS, PaaS (Platform-as-a-Service) and IaaS (Infrastructure-as-a-Service).

A SaaS application is hosted by a provider in the cloud, rented to multiple tenants and accessed by the tenants' users over the Internet [2]. Also, application resources are shared among tenants. In the provisioning of a SaaS application, various stakeholders with different objectives are involved, i.e., providers of all cloud stack layers as well as tenants and their users [1].

Hence, an autonomic and dynamic configuration management is necessary in order to offer these highly configurable SaaS applications.

Some configuration steps, e.g., performed by tenants, are independent from each other. However, others are dependent, e.g., tenant's configuration choices depend on the pre-configuration of the provider. Thus, these later depend on the context-aware of the providers.

In addition, stakeholders' objectives may change over time, e.g., if a tenant decides to change the tenancy contract. Thus, the configuration process needs to support reconfiguration of stakeholder pre-configurations and subsequent ones being further affected.

Our ongoing works are twofold. Firstly, we define context-aware for a configuration management of SaaS applications. Secondly, we suggest an autonomic configuration management based on SPLE (Software Product Line Engineering) [3].

The structure of this paper is as follows. We describe the background in Sections II and III. Then, we show our motivations in Section IV. Section V depicts our futures contributions. In Section VI, we present the related work and the state-of-art. Finally, we conclude this paper in Section VII.

II. VARIABILITY-AWARE SYSTEM

Variability is an ability of software artifacts that allows them to be extended, modified, customized or configured to meet specific needs [4]. In this section, we discuss, in general, the literature concerning systems based on variable modules. Several works have been proposed. We have classified them according to the different phases of software engineering, namely, elicitation time, design time, compile time and binding time. The system variability may occur in all these phases [5].

A. Elicitation Time

It is precisely about managing the variability at the customer's requirements level, examining their priorities and making appropriate choices. A variety of requirement approaches have been proposed in recent works. Barney et al. [6] showed that the management of software product value depends on the context in which the product exists.

B. Design Time

At design time, all variants and variations points are defined in the software architecture or in a complementary feature tree or table. Several approaches were proposed in this phase to model software product lines by using feature models starting with the FODA (Feature Oriented Domain

 ISBN: 978-1-61208-367-4

Analysis) approach [7]. This approach aims at capturing the commonalities and differences points at requirement level.

C. Compilation Time

During the compilation time, the variability described in the architecture must be compiled in the software components (e.g., core assets in a product line) by means of a variety of programming techniques. Cardelli et al. [8] proposed a framework where each module is separately compiled to a self-contained entity and showed that this separation makes it possible to link safely the compatible modules together.

D. Binding Time

Binding time is a property of variation points to delay the design decisions to a later stage, as new requirements or different context conditions may require concretize the variability at any time after design time. Trummer [9] introduced a corresponding data model that is based upon the Café (Cloud Application Framework) model. Applications are composed out of components that may be provisioned separately.

III. CONTEXT-AWARE SYSTEM

An understanding of how context can be used will help us determine what context-aware behaviors to support in our future framework [5].

A. Context

Before specifying our own definition of context to use, we will look at how researchers have defined context in their own work. The first work that introduced the term 'context-aware' was done by Schilit and Theimer [10]. They defined context as location, identities of nearby people and objects, and changes to those objects. Dey et al. [11] defined context as:"*... any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*"

In our work, we will adopt this definition because it allows context to be either explicitly or implicitly indicated by the user.

B. Context-Aware System

The first research investigation of context-aware computing was discussed by Want et al. [12] in 1992. Since then, numerous approaches attempts to define context-aware computing were appeared. Hull et al. [13] defined context-aware computing to be the ability of computing devices to detect and sense, interpret and respond to aspects of a user's local environment and the computing devices themselves. Dey and Abowd [14] defined Context-Aware as:"A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task". In our work, we will adopt this definition because it remains the most generic.

IV. MOTIVATIONS: THE NEED OF AUTONOMIC COMPUTING FOR THE SAAS ACCORDING TO THE TENANT-CONTEXT

SPL have become a common skill for creating software systems that share a common set of commonalities and variabilities that distinguish specific products, thus promoting the development of a family of related products.

Deploying an application in the cloud provides to its owner many advantages: cost reduction, scalability, high availability, etc. However, the migration of an application or the development of a new service in the cloud is not trivial because of the large number of functional and non-functional requirements to deal with [5].

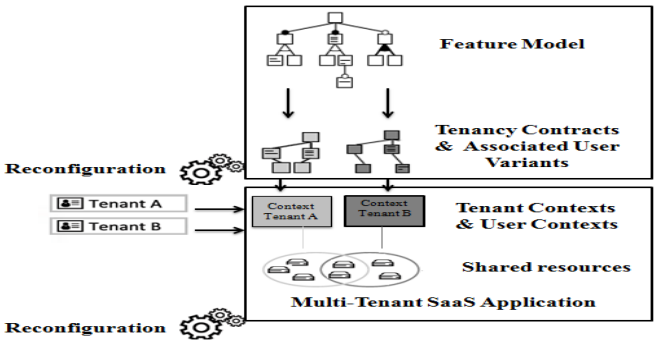


Figure 1. Configuration and instansiation of SaaS application.

We show in Figure 1 how a multi-tenant SaaS application is configured. Tenancy contracts define the provisioned application functionality as well as QoS (Quality of Service) guarantees. Thus, an Extended domain Feature Model (EFM) [15] with attributes is convenient to express this variability and a staged configuration as proposed by Czarnecki et al. is applicable to create those contracts [16]. In contrast to conventional SPL engineering, multiple tenancy contracts and user variants are derived, but integrated into a single application instance in the solution space. To handle this variability, a self-adaptive application architecture was proposed. In this paper, we focus on autonomic managing the variability of SaaS applications by taking into account the context-aware of the system.

V. TOWARD AUTONOMIC CONTEXT-AWRE MANAGEMENT OF VARIABILY

In this section, we will present the notion of autonomic system, and our overview process to achieve autonomic configuration.

A. Autonomic Systems

Autonomic systems are self-regulating, self-healing, self-protecting, and self-improving [17]. Therefore, Autonomic computing capabilities can address the adaptation and reconfiguration challenges of the SaaS cloud layer. Some key open challenges are:

- Self-configuring: As stakeholder objectives change, e.g., if a tenant decides to rent different functionality, the tenant's configuration needs to be reconfigured.

 ISBN: 978-1-61208-367-4

- QoS: Cloud Service Providers (CSPs) need to ensure that sufficient amount of resources is provisioned to ensure that QoS requirements of CSCs (Cloud Service Consumers), such as deadline, response time, and budget constraints are met.
- Security: Achieving security features such as availability. If a coordinated attack is launched against the SaaS provider, the sudden increase in traffic might be wrongly assumed to be legitimate requests and resources would be scaled up to handle them.

B. *Overview of our Process*

Our autonomic system of management variability is presented in Figure 2.

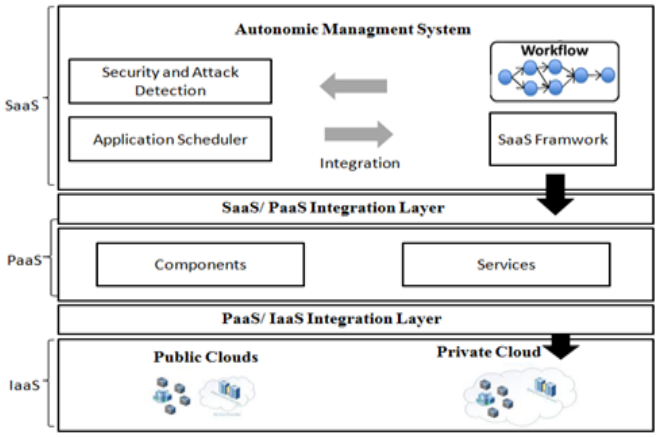


Figure 2. System architecture for autonomic cloud management.

- Application Scheduler: The scheduler is responsible for assigning each task in an application to resources for execution based on user QoS parameters and the overall cost for the service provider.
- Security and Attack Detection: This component implements all the checks to be performed when requests are received .

The workflow of the process proposed which is depicted in Figure 3.

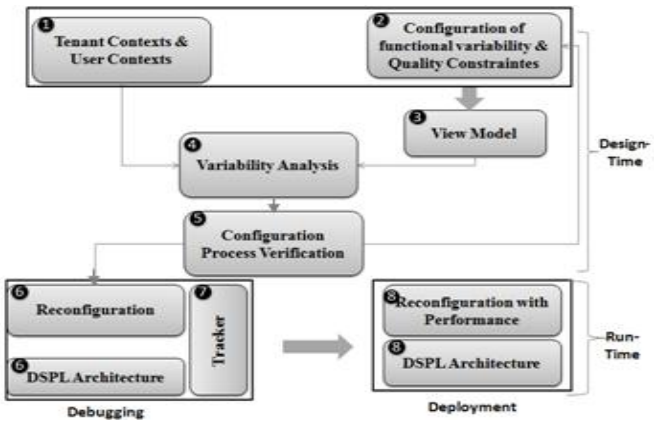


Figure 3.Workflow of our process proposed.

- Step 1: Specifies the context of the reconfigurable system.

User variant configurations are instantiated as user contexts in the SaaS application instance. The users of a tenant have their own user context, each conforming to a user variant configuration. The context of the reconfigurable systems is specified by means of the OWL (Web Ontology Language) [18]. This language provides a vocabulary for describing system context knowledge and for specifying conditions in the context.

- Step 2: Specifies the variability and commonality among functionality and quality properties

The stakeholders have varying requirements on functionality and QoS. Therefore, we need to handle the variability of both. Stakeholders' objectives consider functional variability and variability among quality constraints, e.g., performance, availability, and the server location. We will use an EFM with mixed constraints and group cardinalities.

- Step 3: defines stakeholders and their views on the extended feature model

A stakeholder either represents a person, a member of an organization, or a third party that is involved in the configuration process and has certain concerns regarding the configuration of parts of the EFM. Views are defined by mapping configuration operations specified for the EFM onto groups and categories specified in the View Model. This later defines stakeholders and their views on the extended feature model [19].

- Step 4: Analyzes the reconfigurations before performing them.

Process verification needs to ensure that the configuration process is consistent with the EFM. This is needed for error-correction and avoidance while it would also help users keeping track of their configurations.

- Step 5: Analysis results.

After the given analysis results, the previous configuration can be updated or leveraged at run-time phase.

- Step 6: To Debugs the run-time reconfigurations.

Given the fact that not all potential run-time failures can be anticipated during system design, it is possible to set up MoRE (Model-based Reconfiguration Engine) [20] with a debugging-enabled reconfiguration strategy. This strategy keeps the history of system configurations.

- Step 7: Keeps track of the reconfigurations.

In the context of experimentation, MoRE can store trace entries about the reconfigurations. This provides information

 ISBN: 978-1-61208-367-4

for a posterior analysis, which ranges from context conditions to reconfiguration plans.

- Step 8: To deploy the system in the target platform.

Once the development is finished, there is no interest in debugging information any longer. Therefore, MoRE can be set up with another reconfiguration strategy which lacks debugging support but achieves better performance. We suggest using MoRE featuring a performance-oriented reconfiguration strategy tool.

VI. RELATED WORK

This section presents work that is related to the concepts of our configuration management, which copes with different research fields. Mietzner et al. propose using SPL techniques for configuring multi-tenant SaaS applications [21]. The tenant's configuration decisions are influenced by already deployed services. Concerning our approach, tenants' pre-configurations are not influenced by the configuration of new tenants. Cheng et al. [22] apply SPL techniques on configurable SaaS applications. The description of the application flexibility is created in domain engineering. This catalog is then used to configure the application per tenant. In contrast, we will use EFMs to model the functionality of the application as well as QoS and assume the context-aware of the tenant. Another concept which describes variability for SaaS applications is given by Ruehl et al. [23]. This approach can systematically show variability points and their relationships. This work focuses on the creation of descriptions of variability but not so much on the execution.

Weissbach and Zimmermann [24] tackle the problem of avoiding storing or processing data at undesired location by data-flow analysis. In contrast to our work, this approach is not context-aware. There are also numerous works on context-aware service oriented systems. Du et al. [25] controls data-flow between services to detect malicious services. Context awareness with respect to the client is not assumed. Azeez et al. [26] propose a multi-tenant service-oriented architecture middleware for cloud computing. They;concentrate on multiple users sharing an instance and native multi-tenancy. Contrary to our work, using certain services in context of the location is not considered. Bastida et al. [27] discuss the steps that the service integrators should follow to create context-aware service compositions and also introduce a composition platform that supports the lifecycle of dynamic compositions both at design-time and at runtime. The context part is not explicitly defined in the complete approach.

Table I shows a comparison among several research works in the area of management and configuration of cloud environments. In the state of the art, some work has been performed to combine the benefits SPLE with those of multi-tenancy to facilitate the customization of SaaS applications tailored to the tenant-specific needs. However, none of the current approaches defines explicitly the context-aware of the tenants and users in the complete approach in both design time and run time phase (see Table I). Moreover, it provides no support for context awareness which is one of the keystones for the cloud computing in general and SaaS in particular.

TABLE I. A comparison among research works on Cloud Environment

Research work	Adaptation Type	Phase of system variability	Adaptation Space	Adaptation Mechanisms	Environment
[20]	Dynamic	Design time	Functional	Variability	SaaS
[21]	On-demand	Design time	Functional and non-Functional	Variability	SaaS
[22]	Dynamic	Design time	Functional and non-functional	Variability	SaaS
[23]	Dynamic	Run time	Non-Functional	Variability	Data security in the cloud
[24]	Dynamic	Run time	Non-functional	Variability	IaaS
[25]	Static	Design time and Run time	Functional	variability	Middleware
[26]	Dynamic	Design time and Run time	Functional and context-aware	variability	Composants

VII. CONCLUSION AND FUTURE WORK

This paper presented our first steps towards autonomic and dynamic context-aware configuration variability on the SaaS applications. We identified requirements for a multi-tenant aware SaaS reference architecture at design time as well as at runtime. In addition, we have shown an overview of our process which our framework will be based. We rely

 ISBN: 978-1-61208-367-4

on autonomic system concept in order to allow a dynamic and automatic management of variability for these applications. Furthermore, our dynamic configuration process allows deriving multiple variant configurations that are independent from each other.

Because SPL engineering is a well researched field, we may benefit from developed tools that help to derive valid tenant configurations and we propose to use NSGA-II (Non-Dominated Sorting Genetic Algorithm) algorithms [28] to optimize and select services. Additionally, we plan to take into account context user's evolution. As the cloud market evolves constantly, changes in context can occur that require the application environments to be reconfigured. e.g., a new service is available. To deal with such changes, we propose to adapt evolutionary tree and evolutionary algorithm.

REFERENCES

[1] P. Mell and T. Grance, "The nist definition of cloud computing," NIST Special Publication 800-145, National Institute of Standards and Technology, Information Technology Laboratory, Sept, 2011, pp. 3-8.

[2] G. F. Chong and G. Carraro, "Architecture strategies for catching the long tail," Website, April, 2006, pp. 10-26. [retrieved: 08, 2014]. Available: http://msdn.microsoft.com/en-us/library/aa479069.aspx

[3] M. L. Griss, "Implementing product-line features with component reuse," in Proceedings of the 6th International Conerence on Software Reuse: Advances in Software Reusability, London, UK, June, 2000, pp. 137–152.

[4] D. M. Weiss and C. T. R. Lai, "Software product-line engineering: a family-based software development process," Addison-Wesley Professional, Aug, 1999, 448 pages.

[5] A. Benali and B. El Asri, "Towards dynamic management of variability and configuration of cloud Environments," in press

[6] S. Barney, A. Aurum, and C. Wohlin, "A product management challenge: creating software product value through requirements selection," Journal of Systems Architecture, vol. 54, no 6, June, 2008, pp. 576-593.

[7] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Technical report, CMU/SEI TR-21, USA, Nov, 1990, 148 pages.

[8] L. Cardelli, "Program fragments, linking, and modularization," In Proc. Symp. Principles of Programming Languages (POPL), ACM Press, Mar, 1997, pp. 266–277.

[9] I. Trummer, "Cost-optimal provisioning of cloud applications," Diploma thesis, University of Stuttgart, Faculty of computer science, electrical engineering and information technology, Germany, Feb, 2010, pp. 135 – 142.

[10] B. Schilit and M. Theimer, "Disseminating active map information to mobile hosts," IEEE Network, 8(5), Sept, 1994, pp. 22-32.

[11] A. Dey and G. Abowd, "Towards a better understanding of context and context-awareness," in CHI 2000 Workshop on The What, Who, Where,When, and How of Context-Awareness, nov, 2001, pp. 304-307.

[12] R. Want, A. Hopper, V. Falcao, and J. Gibbons, "The active badge location system," ACM TIS, Jan, 1992, pp. 91-102.

[13] R. Hull, P. Neaves, and J. Bedford-Roberts, "Towards situated computing," International Symposium on Wearable Computers, Oct, 1997, pp. 146-153.

[14] K. Dey and D. Abowd, "Towards a better understanding of context and context-awareness," Georgia Institute of Technology, Atlanta, GA, USA 30332-0280, Sept, 1999, pp. 271-350.

[15] K. Czarnecki, T. Bednasch, P. Unger, and U. Eisenecker, "Generative programming for embedded software: an industrial experience report," In: D.Batory, C.Consel, W.Taha, GPCE 2002. LNCS, vol. 2487, Oct, 2002, Springer, Heidelberg (2002), pp. 156–172.

[16] K. Czarnecki, S. Helsen, and U. Eisenecker, "Staged configuration through specialization and multi-level configuration of feature models," Improvement and Practice Journal, April, 2005, pp. 143-169.

[17] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," Computer, IEEE, Jan, 2003, pp. 41-50.

[18] D. Martin, M. Burstein, J. Hobbs, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara, "OWL-S: Semantic markup for web services," Website, Nov, 2004. [retrieved: 08, 2014].Available: http://www.w3.org/Submission/OWL-S/

[19] J. Schroeter, P. Mucha, M. Muth, K. Jugel, and M. Lochau, "Dynamic configuration management of cloud-based applications," In: SPLC '12:16th International Software Product Line Conference – Vol. 2, ACM, pp. 171–178.

[20] C. Cetina, P. Giner, J. Fons, and V. Pelechano, "Autonomic computing through reuse of variability models at run-time: the case of smart homes," IEEE Computer Society Press, Los Alamitos, CA (2009), Oct, 2009, pp. 37-43.

[21] R. Mietzner, A. Metzger, F. Leymann, and K. Pohl, "Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications," In (PESOS '09) Proceedings, USA, May, 2009, pp. 18-25.

[22] X. Cheng, Y. Shi, and Q. Li, " A multi-tenant oriented performance monitoring, detecting and scheduling architecture based on SLA," In Proceedings of the Joint Conferences on Pervasive Computing, JCPC '09, Dec, 2009, pp. 599-604.

[23] S. T. Ruehl and U. Andelfinger, "Applying software product lines to create customizable software-as-a-service applications," In Proceedings of the 15th International SPL Conference, Volume 2, ACM, Aug, 2011, pp. 16:1-16:4.

[24] M. Weissbach and W. Zimmermann, "Controlling data-flow in the cloud," in The Third International Conference on Cloud Computing, GRIDs, and Virtualization, W. Zimmermann, Y. W. Lee, and Y. Demchenko, Eds. ThinkMind, July, 2012, pp. 24–29.

[25] J. Du, W. Wei, X. Gu, and T. Yu, "Runtest: assuring integrity of data flow processing in cloud computing infrastructures," in Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ser. ASIACCS '10. New York, NY, USA: ACM, Jan, 2010, pp. 293–304. [retrieved: 08, 2014]. Available: http://doi.acm.org/10.1145/1755688.1755724.

[26] A. Azeez, S. Perera, D. Gamage, R. Linton, P. Siriwardana, D. Leelaratne, S. Weerawarana, and P. Fremantle, "Multi-tenant soa middleware for cloud computing," in IEEE CLOUD, July, 2010, pp. 458–465.

[27] L. Bastida, F. J. Nieto, and R. Tola, "Context-aware service composition: a methodology and a case study," In SDSOA '08: Proceedings of the 2nd international workshop on Systems development in SOA environments, New York, NY, USA, ACM, May, 2008, pp. 19–24.

[28] A. Pratap, T. M. K. Deb, and S. Agrawal, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization : NSGA-II," Technical report, Indian Institute of Technology Kanpur, Sep, 2000, pp. 849-858.

.

 ISBN: 978-1-61208-367-4

ASDeDaWaS: An Assistant System for the Design of Data Warehouse Schema

Nouha Arfaoui, Jalel Akaichi
BESTMOD
Higher Institute of Management
Bardo,Tunisia
e-mail: Arfaoui.nouha@yahoo.fr, Jalel.akaichi@isg.rnu.tn

***Abstract*—Data Warehouse has the capacity to integrate data from different data sources for analyses purpose. Despite their importance, many data warehouse projects fail. As cause, we can mention, the poor communication between the developer/designer and the stakeholders, and the bad design that does not respond appropriately to the user requirements. Our work is set in the context of Enterprise Data Warehouse, and we propose a new methodology, Assistant System for the Design of Data Warehouse Schema (ASDeDaWaS). It ensures the design of the schema of the data warehouse taking into consideration the users' requirements and the available data sources, minimizing the computer-scientists intervention.**

Keywords-Data Warehouse Schema; Data Mart Schema; Schema Design; Schema Integration.

I. INTRODUCTION

Data Warehouse (DW) is the "heart of architecture environment and is the foundation of all decision support system processing" [7], since it provides an infrastructure that allows businesses to extract, clean and store vast amount of data. It is defined as "a subject-oriented, integrated, non-volatile, and time-variant collection of data in support of management's decisions" [17].

Concerning the warehousing projects, they are often characterized by their complexity and their huge costs [1] and they may fail during their achievements. According to [1][4][8][9][15], the causes of failure can be summarized as following:

- The nature of those projects requires long periods of development.
- The users' needs are generally poorly expressed by either designers or developers and they are not based on a common terminology.
- The absence of a good design that responds appropriately to the users' requirements.
- The users are, in many cases, not experienced with the technologies of DWs.
- The immaturity and complexity of the design methods and the lack of software tools that support these methods.
- The nonexistence of the right design that ensures the performance today and the scalability tomorrow.

The above difficulties lead to various problems such as the stopping of projects during their implementation, the exceeding of time and/or budget, and the rest.

In order to overcome the previous problems, we propose a new methodology, namely, **A**ssistant **S**ystem for the **De**sign of **Da**ta **Wa**rehouse **S**chema (ASDeDaWaS). It ensures the construction of the schema of the DW incrementally taking into consideration both the users' requirements and the available data sources. It focuses on each department separately, which facilitates the detection and the correction of possible problems and conflicts earlier. It reduces, also, the computer-scientists intervention through the automation of some tasks.

As working hypothesis, it is proposed to present the user requirement as a star schema because it is widely supported by a large number of business intelligence tools; also it has a simple structure, so it is easy to understand the schema. Concerning the data sources, it is proposed to deal with Entity-Relationship (ER) [14] database because it adopts the more natural view that the real world consists of entities and relationships; it incorporates some of the important semantic information and it can achieve a high degree of data independence [14].

As contributions, we propose in this work:

- Using an assistant system to facilitate the collection of users' requirements by exploiting the previous experiences.
- Using a new algorithm to cluster the schemas taking into consideration their semantic aspect.
- Automating the schema integration technique to merge the schemas to generate the logical schemas of the data mart (DM), and the final schema of the DW.

The outline of this work is as following:

- In the second section, we present the state of the art. We summarize some methods that use the mixed approach to design the DW.
- In the third section, we describe our proposed solution and we resume every step.
- In the fourth section, we start by detailing the first step that consists of collecting the users' requirements using an assistant system. The different requirements are modeled as star schemas.
- In the fifth section, the generated schemas are clustered using a new algorithm ak-mode which is an extension of k-mode. It takes into consideration the semantic aspect when clustering the schemas.
- In the sixth section, we propose the application of schema integration technique to ensure the

 ISBN: 978-1-61208-367-4

merging of different schemas existing within every cluster. The proposed technique is composed by schema matching and schema mapping.

- In the seventh section, we propose generating multidimensional schemas from Entity-Relationship (ER) databases.
- In the eighth section, we transform the conceptual schemas that were generated from the users' requirements to logical ones by adding the necessary information extracted from the multidimensional schemas. Using the logical DM schemas, we apply the schema integration technique to build the final schema of the DW.
- We finish this paper with a conclusion and future work.

II. RELATED WORK

Three main approaches have been proposed in the literature to conceive the DW: top-down, bottom-up and mixed.

Top-down starts from the description of the needs of all the users to construct the schema corresponding to the entire DW [5]. According to Ballard et al. [3], this approach has some disadvantages: it is a time-consuming process, it is difficult to collect the different agreement on the data definitions and business rules among all the different workgroups, departments, and lines of business participating. It can delay actual implementation, benefits and return-on-investment and it is length task.

Concerning the bottom-up, the construction of the global schema of DW starts from the different schemas of DMs that are built taking into consideration the requirements of the decision-making users responsible for the corresponding specific business area or process [5]. The problem with this approach is the redundancy and the inconsistency of the data between the DMs [3].

The mixed approach takes advantages of the two previous approaches [5]. It has the speed and the user-orientation of the bottom-up and the integration enforced by a DW in a top-down approach.

In the following, we present some work using the mixed approach to generate the DW and we start by "SelfStar" [6]. The proposed methodology is composed by four steps. It requires human intervention to validate the proposed schema until building the final DW schema. In what follows, we briefly present each step:

- First step: Extracting from the data source, that is expressed using the UML language, the candidate facts and showing them in the intermediate schema. Since the schema of the data source is not easy to be understood by a no-computer scientist user, the system presents a simplified representation automatically extracted from the schema of the source. The user selects the facts and the measures that correspond to his needs. He selects also the operations that will be applied on the measures.
- Second step: Generating the second schema by proposing the different dimensions that can be used with the extracted facts. The extraction of the dimensions is done using: a source described by an exploited schema containing classes and links, and an incomplete decisional base containing one or many facts.
- Third step: Generating a constellation schema (facts, dimensions, and all possible hierarchies). This step generates the candidate hierarchies for each chosen dimension. They are extracted from the classes that are related directly to the dimension using "1..N" link type, and from the attributes of the dimension excluding the attributes having distinct values. The temporal dimension is associated with a standard hierarchy: year, month, date.
- Fourth step: Generating the schema of the DW. In this step the decision makers choose the relevant parameters to their analysis. Then, the system generates the final schema and stores customization metadata for each user to reuse them later by attributing weights to the used classes of the source.

Romero and Abelló [10] propose an approach that uses the end-user information requirements which are expressed as SQL queries and the logical model of the data sources. The final result is a constellation schema. The automatic process is divided into four different stages:

- Concept labeling: It serves to build the multidimensional (MD) graph by applying the labeling standards. For each query, it extracts the MD knowledge.
- Multidimensional graph validation: Each MD-graph that has been generated in the previous step is validated in this stage by generating its multidimensional normal forms.
- Finding representative result: From the previous steps, more than one MD schema can be produced for a given query. Besides, the dimensional data could be considered as an alternative factless fact, although in most cases it will not be relevant to the end-user. This step serves to determine the representativeness of new alternatives and this is done according to some rules. Two sibling graphs differ only in the labeling of one node. Therefore, they have exactly the same labels except for one node, which is considered a factless fact that plays a role in one graph and a strict dimensional role in the other. In short, sibling graphs do not provide new interesting analytical perspectives. They are used to analyze the potential factual data that a dimension may contain. However, in most cases, the end-user would not be interested in this type of analysis.
- Conciliation: It validates each input requirement and generates a potential set of MD schemas for each query. Then, it normalizes MD graphs

 ISBN: 978-1-61208-367-4

Giorgini et al. [13] propose a mixed approach to build the DW. It starts with the requirement analysis that will be mapped next to conceptual level. This step requires the following tasks:

- Organizational modeling: It is centered on stakeholders. It identifies the facts. It is composed by three steps:
  - Goal analysis: Analyzing each goal of each actor in more details.
  - Fact analysis: Determining all the relevant facts and associating goals with facts.
  - Attribute analysis: Determining all the attributes that give a value when facts are recorded.
- Decisional modeling: It focuses on decision makers to extract their information needs. It is composed by four steps:
  - Goal analysis: Identifying the decision makers and establishing the dependencies between them.
  - Fact analysis: Identifying the facts that correspond to different objects of analysis and associating the goals.
  - Dimension analysis: Linking the fact to the dimensions according to the decisional goals of the decision makers.
  - Measure analysis: Associating the measures to each fact previously identified.

Once, they get diagrams that connect enterprise goals to facts, dimensions and measures, they move to the conceptual level by mapping the different elements determined previously. They use two types of frameworks: mixed design framework and demand-driven design framework.

- Mixed design framework: The requirements derived during organizational and decisional modeling are matched with the schema of the operational database in order to generate the conceptual schema for the DW. This is done by performing the requirement mapping, hierarchy construction and refinement.
- Demand-Driven Design Framework: The generation of hierarchies cannot be automatic; here we need the intervention of the designer. Indeed, through his skills and experiences, he can fruitfully interact with the domain experts to capture the existing dependencies between attributes.

Compared to the previous solutions, ASDeDaWaS follows all the necessary steps to generate the schema of the DW. It combines the mixed approach, which generates the logical schemas of the DM from the requirements and the available data sources, and the bottom up approach, which generates the DW schema from the DM logical schemas. Moreover, it offers help using an assistant system to facilitate the collection of the needs reducing the computer-scientists intervention. It can be applied to different departments having different information systems and different needs.

III. OVERVIEW OF ASDeDaWaS

In this section, we describe ASDeDaWaS briefly (Figure 1).

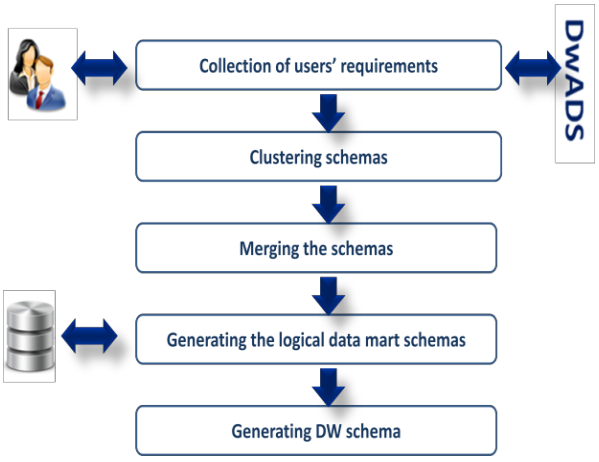


Figure 1. ASDeDaWaS steps.

It starts by collecting the requirements of the different users. It uses an assistant system DwADS (Data warehouse Assistant Design System) to facilitate the specification of the elements basing on the stored traces of the previous users. It defines, then, the possible facts, their measures, and the dimensions with their attributes. From the collected users' requirements, it generates the corresponding schemas that are represented as star.

In the second step, it clusters the different schemas using a new algorithm ak-mode in order to get within one cluster the closest schemas. The new algorithm is an extension of k-mode. It takes into consideration the semantic aspect when making the comparison. Next, for each cluster, it generates the global schema. To achieve this goal, it uses the schema integration technique that is composed by schema matching and schema mapping. The schema matching extracts the semantically closest elements as well as the conflicts and presents them as mapping rules. Using the schema mapping it merges the different schemas to get at the end the global schema. This step allows the generation of conceptual schemas of the DM from the users' requirements. Then, the conceptual schemas are mapped to logical ones. This is done in two steps. In the first one, it extracts all possible multidimensional schemas from the databases. In the second step, it generates the logical schemas. Indeed, it updates the conceptual schemas by adding the necessary information extracted from the multidimensional schemas. Finally, by merging the logical schemas, it builds the final schema of the DW.

IV. COLLECTING OLAP REQUIREMENTS

In order to ensure a good design of DW, it is crucial to start by the requirements that specify "what data should be available and how it should be organized as well as what queries are of interest" [5]. In our case, we need to move

 ISBN: 978-1-61208-367-4

through this step to extract the important objects of the multidimensional schemas (facts, measures, dimensions, and attributes). Despite its importance, not much attention has been paid to this phase causing the failure of 85% of the DW projects to meet business objects, and the no-development of 40% of the DW projects [12].

A. The collection of requirements

To collect the requirements, we give the freedom to the user to express his needs using an easy interface (Figure 2) where he specifies the different objects composing a star schema. This interface uses an assistant system DwADS that helps the user to choose the appropriate objects because the end users may find difficulties to specify their objects [5].

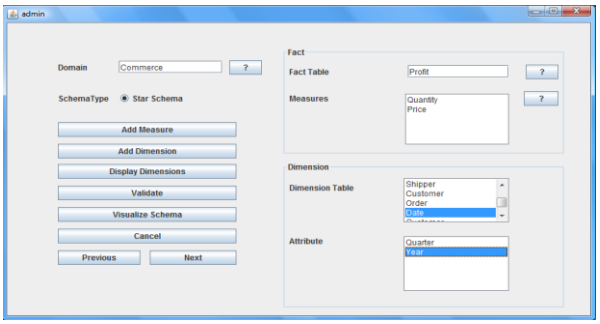


Figure 2. The proposed interface to specify the users' requirements.

Once the user validates his schema, the set of the manipulated objects and the performed actions (add fact, create dimension, and the rest) are stored respecting their order over the time as a trace.

B. The Proposed Assistant System

Our assistant is based mainly on traces. Indeed, it starts by storing the traces of each user during his session, then, it suggests the useful elements after a comparison phase. The system extracts from the trace the objects through the use model and the actions through the observation model. Concerning the comparison step, it uses the episodes to detect the exact position of the user in order to extract next possible objects. This system occurs during the specification of requirements by suggesting to the user the possible elements used to build a first schema basing on the previous experiences.

DwADS performs two main tasks, as presented in Figure 3. The first one corresponds to the building of traces using the use model and the observation model. The second task is about exploiting the previous experiences using the episodes as a method of comparison. This task includes, also, suggestion of the possible next objects to manipulate.

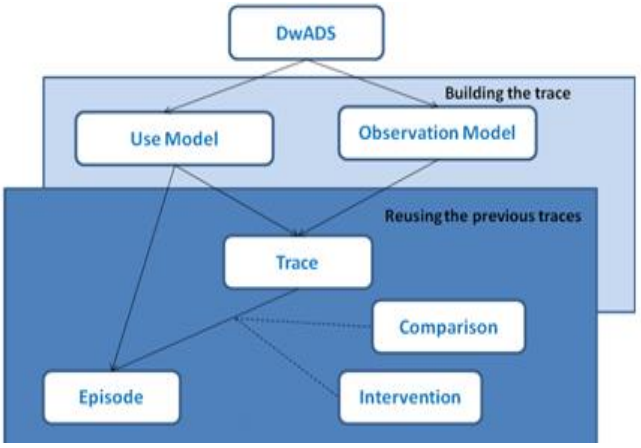


Figure 3. The DwADS composition.

The Use Model (Figure 4): It is used to isolate the objects from the current trace. The objects belong to the following categories (C): "C: Domain", 'C: Model", "C: FactTable", "C: Measure", "C: DimensionTable", "C: DimensionAttribute", and "C: Link". These various categories are linked into single schema through "Contextualization" link. The latter does not present the temporal aspect of the organization of different categories. It, only, shows them connected.

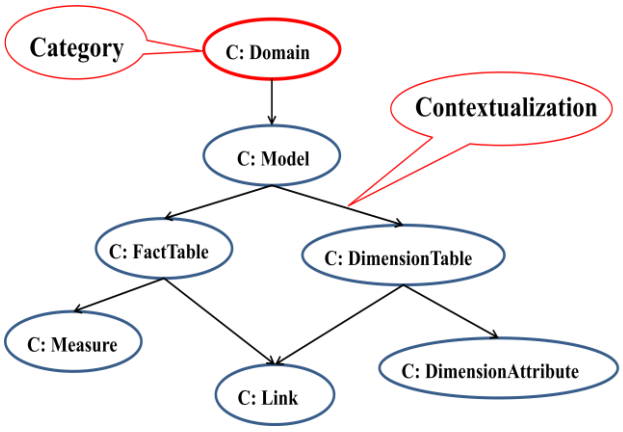


Figure 4. The structure of the use model.

For each requirement specification, the categories are instantiated which gives rise to many possible scenarios. For example the instantiation of "domain" can be "Commerce", "factTable" can be "Transaction", and the rest.

The observation Model (Figure 5): It encapsulates all the actions (||A: ||) handled by a single user during his session. It gives a vision on the use of the application and more precisely on how to deal with the existing objects extracted from the use model.

 ISBN: 978-1-61208-367-4

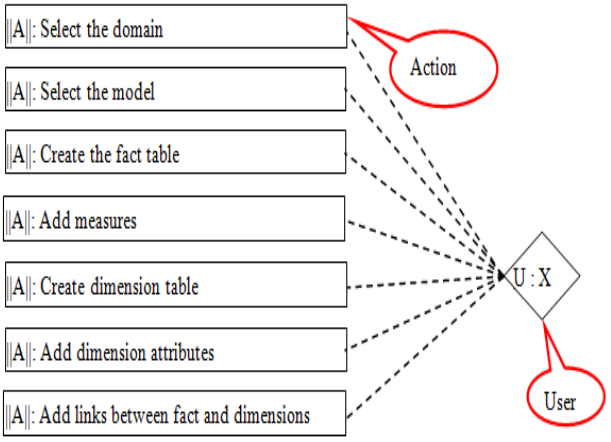


Figure 5. The observation model.

The observation model is instanced once the application is used. It gives different scenarios corresponding to the use of different objects. The scenarios present the actions used to instantiate the objects of the use model.

The trace is a succession of objects and actions over the time. It is built using the use model and the observation model. As example, in Figure 6 we have the trace corresponding to the creation of a star schema having one fact table "Transaction", one measure "Gain" and three dimensions "Customer", "Product" and "Seller", with their attributes over the time.

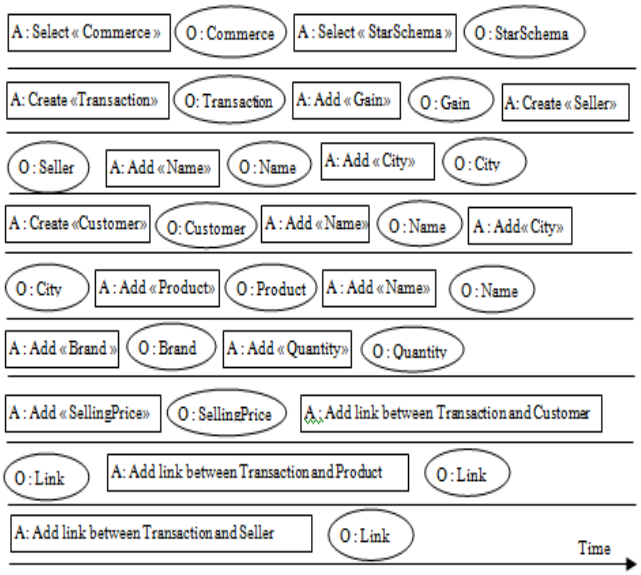


Figure 6. Example of trace corresponding to the construction of star schema.

Exploiting the previous experiences. Each new trace is stored. The set of existing traces in the database are exploited in order to assist the current user by extracting the useful objects. This exploitation is done through performing two tasks. The first one consists of comparing the trace of the current user with the previous traces in order to locate him. The second task is about making the necessary intervention by proposing the possible objects to use.

The comparison: To well exploit the previous traces, it is important to start by locate the user e.g., defining his last manipulated object to be able to predict the next possible objects. The location is done using the episodes that are extracted from the instantiation of the use model. For example, Figure 7 corresponds to three possible instantiation of the category "FactTable" that are "Transactions", "Patient" and "Product".

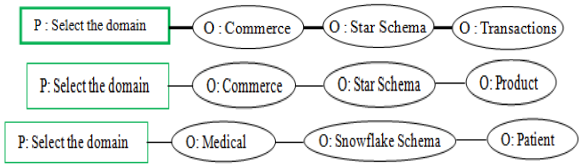


Figure 7. Example of episodes corresponding to the instantiation of the category "FactTable".

Concerning the comparison, there are two cases:

- The system takes into consideration only the last manipulated object, example: "Transaction" (Figure 8).

Figure 8. Example of comparing the last manipulated object.

- The system takes the whole trace into consideration, example: "Commerce, StarSchema, Transaction" (Figure 9).

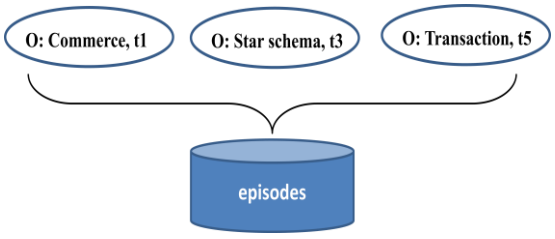


Figure 9. Example of comparing the whole trace respecting the order of objects over the time

.

The intervention: Once the system locates the user, it extracts from the database the set of traces containing the selected object or the set of ordered objects. The

intervention can be done in two different ways:

- The system can intervene by suggesting one next object, example: "Seller", "Country" and "Product" (Figure 10).

Figure 10. Example of intervention by suggesting one possible object.

- The system can intervene by suggesting the rest of the trace respecting the order of the objects, example: "Seller, Name", "Country, City" and "Product, Name" (Figure 11).

Figure 11. Example of intervention by suggestion the rest of trace.

C. *The structure of the generated schema*

At the end of this step, we get a set of schemas corresponding to the users' requirements as example Figure 12.

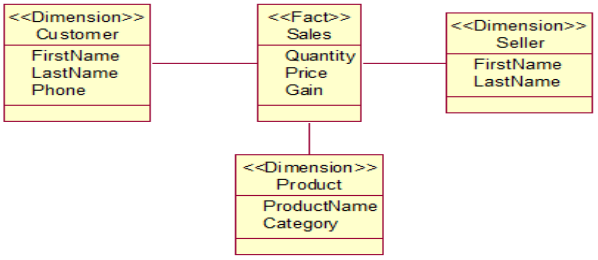


Figure 12. Example of user requirement presented as a star schema.

Each one is represented as star schema having the following structure:

- The fact table corresponds to the subject of analysis. It is defined by a tuple: **FN** and **MF { }** with:
  - **FN**: represents the name of the fact. e.g., "Sales"
  - **MF {m1, m2, m3, m4, …}**: corresponds to the set of measures related to the fact F, e.g., "Quantity, Price and Gain".
- The dimension tables represent the axis of analysis. Each one is composed by: **DN** and **A{ }** with:
  - **DN**: corresponds to the dimension name, e.g., "Customer, Seller and Product".
  - **A {a1, a2, a3, a4, …}:** presents the set of attributes describing the current dimension D, e.g., for the dimension "Customer" the attributes are "FirstName, LastName and Phone".

V. CLUSTERING OLAP REQUIREMENTS SCHEMAS

At the end of the previous step, we get a set of schemas corresponding to the different requirements. In order to exploit them, we propose their clustering according to their domain using a new algorithm ak-mode that takes the semantics aspect into consideration.

Clustering is the unsupervised classification of patterns into groups called Clusters [2]. It involves dividing a set of data points into non-overlapping groups, or cluster of points [16], and this is exactly what we aim to do with OLAP Requirement Schemas (ORSs), i.e., grouping them with maximizing their similarity within one cluster and minimizing it between clusters.

The clustering proposes different algorithms. To choose the appropriate one, we compare them in term of "time complexity", as presented in Table I.

TABLE I. CLUSTER ALGORITHMS FOR CATEGORICAL DATA.

Algorithm	Complexity	Coefficient
K-MODE	O (n)	Simple Matching
ROCK	$O(kn^2)$	Links
QROCK	$O(n^2)$	Threshold
COOLCAT	$O(n^2)$	Entropy
LIMBO	O (nLogn)	Information Bottleneck
MULIC	$O(n^2)$	Hamming measure

We can notice that the k-mode has O (n), which is the lowest complexity, cannot deal with our schemas because it does not take into consideration the semantic aspect of the elements; so, we extend it and we propose aK-Mode.

A. *The Extension of Simple Matching Dissimilarity Measure*

Let Sch1 and Sch2 be two schemas belonging to the same cluster.

Let Ci be the categories of elements existing in the schema with Ci = {fact, dimension, measure, attribute}.

When we calculate the similarity between the elements of the two schemas, we should take into consideration the following points:

 ISBN: 978-1-61208-367-4

- The identical: We use the same elements name in the two schemas.
  DeId (ei, ej) = 1 if ei and ej are identical and 0 if not.
- The synonymous: We use two different names that have the same meaning.
  DeSy (ei, ej) = 1 if ei and ej are synonymous, and 0 if not.
- The typos: We make mistakes when writing the name of the element. In this case, we calculate the degree of error. If it is low, we are in the case of typing error. If it is high we are in the case of two different words. In the following we only take into consideration the first case.
  DeTy (ei, ej) = 1 if ei and ej are the same with the existence of typing error, 0 if not.
- The post-fixe: We use post- fixes to design the same thing.
  DePost (ei, ej) = 1 if one the two elements is the post-fixe of the other, and 0 if not.
- The pre- fixe: We use pre-fixes to design the same thing.
  DePre (ei, ej) = 1 if one of the elements is the pre-fixe of the other, and 0 if not.

The degree of similarity between ei and ej (DeSim (ei, ej)) is measured by the numeric value {0} or {1}, and it is calculated as following formula (1):

$$DeSim(ei, ej): Sch1 \times Sch2 \rightarrow \{0, 1\}$$
$$DeSim(ei, ej) = [DeId(ei, ej) + DeSy(ei, ej) + DeTy(ei, ej) + DePost(ei, ej) + DePre(ei, ej)] \quad (1)$$

The new formula (2) of the simple matching (SM) dissimilarity measure is defined as following:

$$Coef_{SM}(sch1, sch2) = [(MaxD - CoefD) / MaxD] + [(MaxM - CoefM) / MaxM] + [(MaxF - CoefF) / MaxF] + [(MaxA - CoefA) / MaxA] \quad (2)$$

With:

- MaxD: It is the maximum number of dimensions existing in the two schemas.
- CoefD: It is the number of similar dimensions existing in the schemas using "DeSim".
- MaxM: It is the maximum number of measures existing in the two schemas.
- CoefM: It is the number of similar measures existing in the schemas using "DeSim"
- MaxF: It is the maximum number of facts existing in the two schemas.
- CoefF: It is the number of similar facts existing in the schemas using "DeSim"
- MaxA: It is the maximum number of attributes existing in the two schemas.
- CoefA: It is the number of similar attributes existing in the schemas using "DeSim"

B. The ak-Mode Algorithm

The algorithm of aK-mode is described as following:

a) Define the 'k' number of existing domains.

b) Select 'k' initial modes. The initial modes correspond to the schemas that were selected randomly from each cluster.

c) Allocate a schema to the cluster whose mode is the nearest to the cluster, using the formula (2).
Update the mode of the cluster after each allocation.

d) After all schemas have been allocated to the respective cluster, retest the schemas with new modes and update the clusters.

e) Repeat steps (b) and (c) until there is no change in clusters.

VI. MERGING THE USERS' REQUIREMENTS SCHEMAS

In this part, we generate the schemas of the DM from the existing clusters using the schema integration technique that combines the matching and the mapping. Compared to the others, our methodology does not require the pre-integration phase since the used schemas have the same model that was unified from the beginning.

A. Schema Matching

The schema matching is considered as one of the basic operations required by the process of data integration [11]. It is used to solve the problem related to the heterogeneity of the data sources by finding semantic correspondence between the elements of the two schemas. This phase is iterative; it takes two schemas as input each time to get as output a set of mapping rules in order to facilitate the merging task in the next step.

To ensure the effective schema matching, we focus on linguistic matching of names of schemas' elements and according to Li et al. [18], it proceeds in three steps: normalization, categorization and comparison.

- Normalization: Different names design the same thing but they are written differently. They perform tokenization (e.g., parsing names into tokens based on punctuation, case, and the rest), expansion (identification of the abbreviation, acronyms, and the rest). So, we propose the use domain ontology, lenvenshtein name, and the rest.
- Categorization: It is to group the elements composing the schemas by categories: fact, dimension, measures, and attributes to reduce the number of one-to-one comparison eliminating the unnecessary comparisons.
- Comparison: A coefficient of linguistic similarity is calculated by comparing the tokens extracted from the names of the elements using the formula (1).

 ISBN: 978-1-61208-367-4

B. Schema Matching Steps

The schema matching serves to extract the mapping rules that will be used to facilitate the merging of schemas. Our proposed methodology is composed by the following steps:

- Categorization: It is to specify the category of each element. This can reduce the risk of error which provides a gain of time.
- Construction of the similarity matrix: It is about using a similarity matrix to find the closest elements. The cells contain the coefficient of similarity of the different elements belonging to the same category using the formula (1).
- Generation of the mapping rules: The rules visualize the conditional relationships between the instances of the categories. They are expressed as: **"If** Similar (X, Y) **then** Action (X, Y)**"**, with:
  - X and Y belong to the same category (fact, measure, dimension, or attribute).
  - Similar ( ): It is a function that specifies if the two inputs are similar or not.
  - Action ( ): It specifies the actions to perform. They can be union, or intersection.

The different rules are stored into rules database.

C. Schema Mapping

Once we extract the mapping rules; we move to the next where we apply those rules to merge the schemas. The schema mapping is a qua-triple $M = (sch1; sch2; T; \delta)$. "sch1" is the first schema, "sch2" is the second schema, "T" is the target schema, and "δ" is a set of formulas over <sch1, sch2; T>.

An instance of M is an instance of $<s_1, s_2; t; \delta_i>$ over <sch1, sch2; T; δ> that has a specific formula in the set δ_i. The formulas existing in δ_i correspond to one of the following functions:

- Union: R = union (ei, ej) implies that R contains all the components of "ei" and all components of "ej". It is applied when the two elements are identical.
- Intersection: R= intersection (ei, ej) implies that R contains the components that exist in "ei" and "ej". It is applied when the two elements are equivalent and not identical.

VII. GENERATING MULTIDIMENSIONAL SCHEMAS FROM DATABASE

In this section, we propose an algorithm to generate all multidimensional schemas from the data sources. This helps to construct the DM logical schemas by making the necessary modifications. We suggest working with Entity-Relationship (ER) model of the data source.

Our algorithm starts by extracting the potential facts and dimensions. For each fact, it extracts all possible measures. For each dimension, it adds the attributes.

Step 1: Normalize the ER model

Apply the 1NF, 2NF and 3NF to construct the ER normalized:

- First Normal Form (1NF): It is that there should be no nesting or repeating groups in a table.
- Second Normal Form (2NF): It is that the key attributes determine all non-key attributes.
- Third Normal Form (3NF): It is that the non-key attributes should be independent.

Step 2: Build the tree from ER model

From the ER model, we extract the entities (**Ef**) having n-ary relationships with other entities and those having numerical attributes. They represent the potential facts. Every **Ef** becomes to root of the tree. The number of trees corresponds to the number of **Ef** entities. From the ER, we extract the entities (**E**) that are directly linked to **Ef** corresponding to the potential dimensions.

Step 3: Transform the tree to multidimensional model

- The root of each tree becomes the fact table.
- The existing numeric attributes become the potential measures
- The measures are defined by an aggregation functions that are specified by the user.
- The nodes that are directly linked to the roots are transformed to dimensions keeping their attributes and their primary keys.
- The primary keys of the children nodes become foreign keys in the parents' nodes.

Figure 13 presents an example of multidimensional schema.

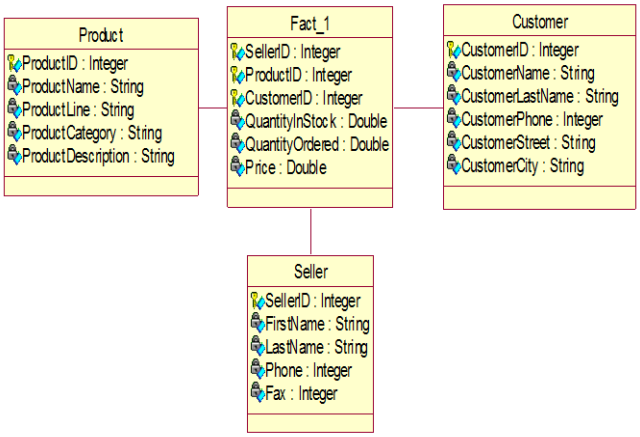


Figure 13. Example of multidimensional schema generated from an ER database.

It is composed by one fact table "Fact_1", three measures "QuantityInStock, QuantityOrdered, Price", set of keys defining the primary key of the fact table, three dimensions "Product, Customer, Seller". Each dimension has its primary key and a set of attributes.

 ISBN: 978-1-61208-367-4

VIII. THE DATA WAREHOUSE SCHEMA

In this section, we generate the schema of the DW. To realize this task, we need first to generate the logical schemas of the DM.

A. Generating the Data Mart Logical Schemas

The purpose of this step is to move from the conceptual schemas to the logical ones. At this level, we have two types of schemas. The first ones were generated from the requirements, they correspond to the **D**ata **M**art **U**ser **S**chemas (DMUS)s and they are modeled as star. The second ones were generated from the different databases, they correspond to the **D**ata **M**art **M**ultidimensional **S**chemas (DMMS)s and they are modeled as star schemas. The validation of DMUS is about adjusting the needs with databases so that we have the source from which we can extract data later.

In order to achieve this task, we compare the two types of schemas to extract the closest ones, then, we update the DMUS by adding the necessary information.

To compare the DM schemas, we start by classifying their elements into the following categories: fact, measure, dimension, and attribute. Using the similarity matrix, we extract the closest schemas. The updating task has as purpose transforming the conceptual schema to logical one. To achieve this task, we need human intervention. Indeed, we present the elements of two types of schemas and the final user specifies the necessary elements to keep. For example, Table II visualizes the elements extracted from Figure 12 and those extracted from Figure 13 to specify the elements of the final schema. For example, the attribute "FirstName", extracted from the conceptual schema, has its corresponding attribute existing in the multidimensional schema. It has as type "String". This attribute is added to the final schema with its type. The same process is applied to the rest of elements.

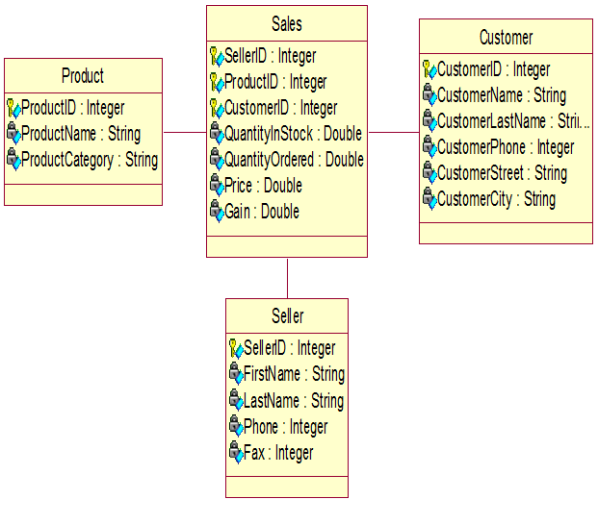


Figure 14. The logical schema of the Data Mart.

Figure 14 corresponds to the logical schema of the DM once the conceptual schema is updated.

B. Generating the Data Warehouse Schema

At this level, we have a set of logical DM schemas. To generate the final schema of the DW, we propose the use of schema integration technique as presented previously. It is composed by schema matching and schema mapping. This process is iterative. It takes every time two schemas as input. We stop when we get one final schema.

IX. CONCLUSION AND FUTURE WORK

The DW has the capacity to integrate huge amount of historical data for analysis purpose. It plays an important role with organizations. Despite their importance, many projects fail because of the absence of good design.

In this work, we proposed a new methodology to design the schema of the DW reducing the computer-scientists intervention. It takes into consideration the needs of each department separately to facilitate the detection of possible problems earlier, as well as existing databases to get at the end the best schema. Indeed, it starts by collecting the users' requirements using an assistant system that exploits the previous experiences. Then, it clusters them using ak-mode to build first DM schemas that are generated using the schema integration technique. Besides, it revises those schemas to generate the logical DM schemas that serve at the end to build the final DW schema.

As future work, we propose dealing with other kind of data sources (UML, XML files, and the rest). We propose, also, taking into consideration the evolution of the schema of the database.

REFERENCES

[1] A. N. AbuAli, and H. Y. Abu-Addose, "Data Warehouse Critical Success Factors", European Journal of Scientific Research, vol. 42 pp.326-335, 2010.

[2] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data Clustering: A Review", ACM Computing Surveys, vol. 31, 1999, pp. 264-323.

[3] C. Ballard, D. Herreman, D. Schau, R. Bell, E. Kim, and A. Valencic, "Data Modeling Techniques for Data Warehousing", International Technical Support Organization, 1998.

[4] E. Börger, "High Level System Design and Analysis using Abstract State Machines", FM-Trends, 1998, pp. 1-43.

[5] E. Malinowski, and E. Zimanyi, Advanced Data Warehouse Design, From Conventional to Spatial and Temporal Applications, Springer Verlag Berlin Heidelberg, 2008.

[6] F. Abdelhédi, F. Ravat, O. Teste, and G. Zurfluh, "SelfStar: an interactive system for the construction of multidimensional schemas", Informatique des Organisations et Systèmes d'Information et de Décision (INFORSID), pp. 335-350, 2011.

[7] H. R. Nemati, D. M. Steiger, L. S. Iyer, and R. T. Herschel, "Knowledge warehouse: An architectural integration of knowledge management, decision support, artificial intelligence and data warehousing". Decision Support Systems, vol. 33, Jun. 2002, pp. 143–16.

[8] M. Golfarelli, "From User Requirements to Conceptual Design in Data Warehouse Design". In Data Warehousing Design and Advanced Engineering Applications Methods for Complex Construction, IGI Global, Hershey, 2009, pp. 1-16.

[9] M. Golfarelli, and S. Rizzi, "WAND: A CASE Tool for Data

 ISBN: 978-1-61208-367-4

Warehouse Design". In Demo Proceedings of 17th International Conference on Data Engineering (ICDE), pp. 7-9, 2010.

[10] O. Romero, and A. Abelló, "Automatic validation of requirements to support multidimensional design". Data Knowledge Engineering, vol. 69, pp. 917-942, 2010.

[11] P. A. Bernstein, and S. Melnik, "Meta data management". In Proceedings of the IEEE CS International Conference on Data Engineering. IEEE Computer Society, 2004.

[12] P. Andritsos, P. Tsaparas, R. J. Miller, and K. C. Sevcik, "LIMBO: Scalable Clustering of Categorical Data". In Proceedings of the 9th International Conference on Extending Database Technology (EDBT), 2004, pp. 123-146.

[13] P. Giorgini, S. Rizzi, and M. Garzetti, "Goal- oriented requirement analysis for data warehouse design". In Proceedings of the 8th International Workshop on Data Warehousing and OLAP (DOLAP), 2005, pp. 47 - 56 .

[14] P. P. S. Chen, "The Entity-Relationship Model-Toward a Unified View of Data". ACM Transactions on Database Systems, vol. 1, 1976, pp.9-36.

[15] S. R. Gardner, "Building the Data Warehouse". Communications of the ACM, Vol.41, 1998, pp. 52-60.

[16] V. Faber, "Clustering and the Continuous k-means Algorithm". Los Alamos Science, vol. 22, 1994, pp. 138-144.

[17] W. H. Inmon, "Building the Data Warehouse. John Wiley & Sons Inc, 2005.

[18] Y. Li, D. Liu, and W. Zhang, "A Generic Algorithm for Heterogeneous Schema Matching". International Journal of Information Technology, vol. 11, 2005, pp. 36-43.

TABLE II. TRANSFORMATION OF THE CONCEPTUAL SCHEMA TO LOGICAL ONE.

Category	**Conceptual level**	**Multidimensional level**		**Logical level**	
		Element	**Type**	**Element**	**Type**
Fact	Sales	Fact_1	-	Sales	-
FactKey	-	SellerID	Integer	SellerID	Integer
	-	ProductID	Integer	ProductID	Integer
	-	CustomerID	Integer	CustomerID	Integer
Measure	Quantity	QuantityInStock	Double	QuantityInStock	Double
		QuantityOrdered	Double	QuantityOrdered	Double
	Price	Price	Double	Price	Double
	Gain	-	-	Gain	Double
Dimension	Customer	Customer	-	Customer	-
DimensionKey	-	CustomerID	Integer	CustomerID	Integer
Attribute	FirstName	CustomerName	String	CustomerName	String
	LastName	CustomerLastName	String	CustomerLastName	String
	Phone	CustomerPhone	Integer	CustomerPhone	Integer
	-	CustomerStreet	String	CustomerStreet	String
	-	CustomerCity	String	CustomerCity	String
Dimension	Seller	Seller	-	Seller	-
DimensionKey	-	SellerID	Integer	SellerID	Integer
Attribute	FirstName	FirstName	String	FirstName	String
	LastName	LastName	String	LastName	String
	-	Phone	Integer	Phone	Integer
	-	Fax	Integer	Fax	Integer
Dimension	Product	Product	-	Product	-
DimensionKey	-	ProductID	Integer	ProductID	Integer
Attribute	ProductName	ProductName	String	ProductName	String
	Category	ProductCategory	String	ProductCategory	String
	-	ProductLine	String	-	-
	-	ProductDescription	String	-	-

Towards Implementation and Design of Multi-tenant SaaS Based on Variabiliy Management Mechanisms

Houda Kriouile, Bouchra El Asri, M'barek El Haloui and Asmae Benali
IMS Team, SIME Laboratory
ENSIAS, Mohammed V University
Rabat, Morocco

Email: [houda.kriouile, mbarek.haloui, asmae.benali]@um5s.net.ma,
elasri@ensias.ma

Abstract—**Software as a Service (SaaS) is a form of Cloud Computing that involves offering software services on-line and on-demand via Internet deemed a main delivery support. Multi-tenancy is a tool to exploit economies of scale widely promoted by SaaS model. Even so, the ability of a SaaS application to be adapted to individual tenant's needs seem to be a major requirement. Thus, in this paper we introduce an approach proposing a more flexible and reusable SaaS system for Multi-tenant SaaS application. The approach introduced is based on integrating a deployment variability that enables the customers to choose with which others tenants they want or do not want to share instances with a functional variability using Rich-Variant Components.**

Keywords-SaaS; Rich-Variant Component; Functional Variability; Deployment Variability; Multi-tenancy.

I. INTRODUCTION

With the age of Cloud Computing, several forms of Cloud services have emerged thanks to the Internet services development and the customers' needs evolution, in particularly the Software as a Service (SaaS) form. The latter refers to software distribution model in which applications are hosted by a service provider and made availability to customers over a network, typically the Internet. A key enabler in Cloud Computing to exploit economies of scale is the multi-tenancy, a notion of sharing resources among a large group of customer organizations, called tenants. But, the multi-tenant application responds only to needs that are common to all tenants. So, a plethora of work research has been performed to facilitate SaaS applications customization according to the tenant-specific requirements by exploiting benefits of both variability management and multi-tenancy [1][2][3]. In the same vein, we aim to create a flexible and reusable environment enabling greater flexibility and suppleness for customers while leveraging the economies of scale. For this purpose, we propose a solution integrating a functional variability at application components level and a deployment variability at multi-tenants level as well.

This paper is divided into five main sections along with this introduction. Section II provides an overview on variability management mechanisms, Cloud Computing and Multi-tenancy as a background concepts for our work research, then deals with the incentives and motivations for the proposed approach. Section III presents several approaches studied as related work and positions our contribution. Section IV initiates our contribution which consists on integrating functional and deployment variabilities for SaaS applications. Section V provides some outstanding of our approach and future works. Finally, Section VI is a conclusion of the paper.

II. BACKGOUND AND MOTIVATION

In the following subsection, some variability management mechanisms are presented, followed by a short introduction to the Cloud Computing and the Multi-tenancy notions as a background for our work. Finally, the motivation of our contribution consisting on the need of managing variability for Cloud environment is discussed.

A. Variability managment mechanisms

Variability is the ability of a software artifact to be adapted for a specific context [4]. For example, it could be the ability to be extended, configured, customized or modified. A request for change requires the evolution of systems. Therefore, the variability of the system or the software must be managed during all lifecycle's phases (e.g., the specification phase, the conception phase, the testing phase, the implementation phase, etc.).

A variety of mechanisms and approaches are proposed for the management of system's variability. These mechanisms intervene at the level of the different lifecycle's phases. Examples are cited bellow:

- Specification phase: Iqbal, Zaidi and Murtaza propose a model for requirement prioritization using Analytical Hierarchical Process (AHP) [5].
- Conception phase: Several approaches were proposed in this phase to model software product lines by using feature models as the Feature Oriented Domain Analysis (FODA) approach [6], which aims to capture the commonalities and the points of difference at requirement level. Many other approaches provided extensions to the FODA approach. Such as the Feature-Oriented Reuse Method (FORM) [7], whose main contribution is the decomposition of the feature model layers to describe different perspectives (capacity, environment, technology).
- Testing phase: Erwing and Walkingshaw propose organizing the space of all variations by dimensions, which provides scoping and structuring choices [8]. They consider the concept of "variation programming" for a flexible construction and transformation of all kinds of variation structures [8].

 ISBN: 978-1-61208-367-4

- Implementation phase: Trummer proposed a corresponding data model [9] that is based upon the Composite Application Framework (Cafe) model [1]. Applications are composed out of components that may be provisioned separately.

From the cited works above, the interest of variability management mechanisms is evident. Particularly, these mechanisms are useful for managing the functional variability and the deployment variability into Cloud environment, specially for Multi-tenant SaaS applications.

B. The Cloud Computing and Multi-tenancy

Cloud Computing as defined by the National Institute of Standards and Technology (NIST) is the access via a telecommunications network, on demand and self-service, to a shared pool of configurable computing resources [10]. Cloud Computing is the use of computing resources - hardware and software - that are provided as a service over a network, usually the Internet. Cloud Computing entrusts remote services with a user's data, software and computation [10].

Our work focuses on Cloud Computing Services from the kind of Software as a Service (SaaS). In this type of service, applications are made available to consumers. Applications can be manipulated using a web browser. As a tool to exploit economies of scale, SaaS promotes Multi-tenancy [3].

Multi-tenancy is the notion of sharing resources among a large group of customer organizations, called tenants. That is, a single application instance serves multiple customers. But, even though multiple customers use the same instance that each of which has the impression that the instance is designated only to them. This is achieved by isolating the tenants' data from each other. Compared to single-tenancy, Multi-tenancy has the advantage that infrastructure may be used most efficiently as it is feasible to host as many tenants as possible on the same instance. Thus, maintenance and operational cost of the application decreases [3]. In Multi-tenant SaaS applications, the variability may have fundamentally different sources (evolution, maintenance, tenant's requirements, etc.), but is naturally present [2].

C. Motivation: On the need of managing variability for the Cloud Environment

The emergence of Cloud Computing has necessitated more and more variability in the form of types of services, types of deployment, and the different roles of Cloud participant. Thus, variability modeling is required to manage the inherent complexity of Cloud systems.

SaaS application are consumed by many customers that have different requirements. Thus, customers that consume the same application usually exhibit varying requirements needs. Varying requirements usually necessitate varying software architectures. In other words, when applications' requirements are changed, the software architectures of these applications are modified to satisfy the changed requirements. Therefore, both requirements and architectures have intrinsic variability characteristics.

Moreover, other concerns are raised by Multi-tenancy. For example the need to ensure the correctness of all possible configuration of the application. It is not sufficient to guarantee the correctness of a single application's configuration.

On the other hand, in Multi-tenant SaaS application, the consumer does not have to worry about making updates and upgrades, adding security and system patches and ensuring service availability and performance. In addition to that, the rapid elasticity and the resource pooling are essential Cloud characteristics [10], which promote variability for Cloud Computing environment and particularly for Multi-tenant contexts.

III. RELATED WORK

Several research works have been performed in the context of architectural patterns for developing and deploying customizable Multi-tenant applications for Cloud environment. Fehling and Mietzner propose the Composite-as-a-Service (CaaS) model [11]. They show how applications built of components, using different Cloud service models, can be composed to form new applications that can be offered as a new service. These applications have been designed in the spirit of customization, thus their variability was modeled using the application model and variability model from the Cafe Framework [1], which allows exploiting economies of scale by the use of highly flexible templates enabling increasing customers base. Our work aims to exploit economies of scale from two sides by the use of Multi-tenancy and the introduction of the new concept of Multiview, that has not been used in any of the related work studied.

In the context of the Late Binding Service - which enables service loose coupling by allowing service consumers to dynamically identify services at runtime - Zaremba, Vitvar, Bhiri, Derguech and Gao present models of Expressive Search Requests and Service Offer Descriptions allowing matchmaking of highly configurable services that are dynamic and depend on request [12]. This approach can be applied to several types of services. In the remainder of their work, Zaremba, Bhiri, Vitvar and Hauswirth apply their approach [12] on the domain of Cloud Computing, more exactly on the IaaS services that are highly configurable, change dynamically and depend on requests [13]. This approach deals with a different area of cloud application which is IaaS services. Moreover, this approach does not propose a solution to exploit economies of scale and only deals with one type of variability, which is the deployment variability.

Ruehl, Wache and Verclas address the deployment variability based on the SaaS tenants requirements about sharing infrastructure, application codes or data with other tenants [3]. They propose a hybrid solution between Multi-tenancy and simple tenancy, called the mixed tenancy. The purpose of this approach is to allow the exploitation of economies of scale while avoiding the problem of customers hesitation to share with other tenants [3]. Authors focus on the deployment variability and neglect the functional variability management.

In [2], an integrated service engineering method, called service line engineering, is presented. This method supports co-existing tenant-specific configurations and that facilitates the development and management of customizable, Multi-tenant SaaS applications without compromising scalability [2]. In contrast to our approach, this method - as well as the other approaches cited - does not address to the accessibility by roles, which is allowed in our work by the use of Multiview concept.

 ISBN: 978-1-61208-367-4

The Multiview notion allows applications to dynamically change the behavior according to the enabled user's role or viewpoint.

The next section deals with the initiation and the explanation of the approach subject of our contribution, consists of integrating the functional variability with the deployment variability for Multi-tenant SaaS applications.

IV. TOWARDS THE IMPLEMENTATION AND THE DESIGN OF MULTI-TENANCY SAAS

One of the main focuses of our team research is to work on complex systems variability according to functional areas that have initiated the concept of Multiview Component [14]. The Multiview Component concept is a component model for viewpoint in the perspective of View-based Unified Modeling Language (VUML) approach [15].

Our work aims to define a way to design and descript capabilities and variability of Rich-Variant services. In this intention, our contribution is to establish a flexible and reusable SaaS environment while exploiting economies of scale. That is, our work in progress consists of providing Multi-tenant applications based on Rich-Variant Components (RVC), which allow customers to choose among other tenants who they want or do not want to share the deployment with. This implies that these composite SaaS applications are made up of a number of RVC components; each one of which provides an atomic functionality and modifies their behavior dynamically depending on the Multi-tenant variability.

A glimpse of our architectural vision for the approach is provided in Figure 1. All tenants use the same execution engine that executes tenants' specific configurations by communicating with a Web server. A tenant is a customer who pays to use the application. It could be an enterprise, a company or any kind of organization wishing to rent the application.

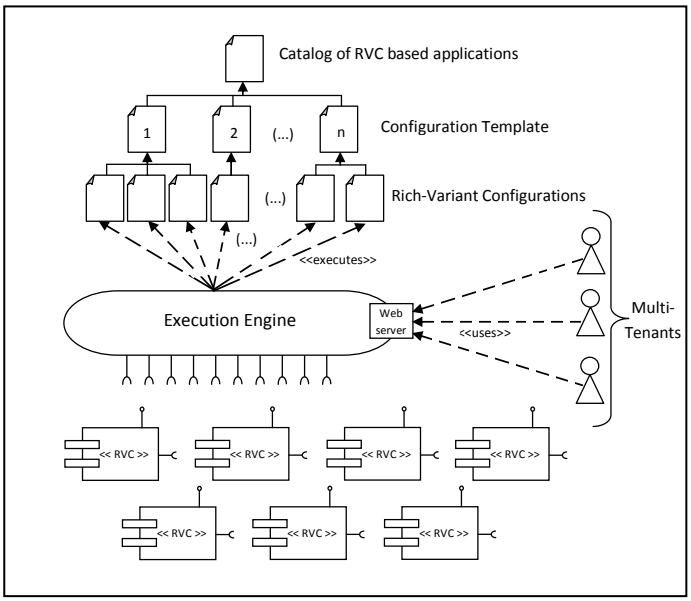


Figure 1. The architectural vision

Each tenant has several users who are actually their employees. These end-users will be categorized according to their business and needs to form different roles or viewpoints. So, applications which are based on RVC components behave differently and this is according to the enabled role or viewpoint thereof, as it was mentioned earlier.

The *catalog* is a formal description of all the applications offered by a provider. It describes the functional variability of each application and specifies the variability points of an application to show how it could be customized.

For each application, the *configuration template* describes the different RVC components that must be linked to create the specific application. The configuration template contains instantiations of the catalog related to the application. Thus, the variability points of each RVC component that require specific tenants information are not configured yet at this level.

Based on a particular configuration template, the *Rich-Variant configuration* describes a specific application tailored for a specific tenant with a dynamic behavior changing during the execution according to the end-users' enabled role or viewpoint. In addition, the parameters or variability points provided by each RVC component are defined at the Rich-Variant configuration level.

From the catalog, the Multi-tenants choose the features and functionalities they need and specify the necessary parameters to obtain their specific Rich-Variant configuration. This fact enables the functional variability. Besides, the use of RVC components allows more flexibility according to end-users business.

In a further work, we plan to define a number of deployment models. Namely a Public deployment model to enable sharing deployment with all other tenants. A Private deployment model to not enable sharing deployment with any other tenant. And a Hybrid deployment model to enable specifying with who share and who do not share the component's instance. For each RVC component, the tenant chooses one deployment model. Thus, we intend to allow the variability of deployment by permitting customers to choose with whom they want to share an instance of a particular RVC component based on the aforementioned deployment models.

Our work has been implemented on a case study to demonstrate the value and feasibility of the approach. The case study consists of a SaaS application for managing private schools, accessible from a web browser. Schools which are tenants of the application benefit, undoubtedly, from deployment variability and functional variability in a flexible, reusable and dynamic environment according to the different needs of the end-users (e.g., administrator, professor, student, etc.).

V. OUTSTANDING AND FUTURE WORKS

In our research work, we seek to integrate both functional and deployment variability. Also, we look to improve reusability by the use of RVC components. In addition, our approach enables flexibility according to the tenants' requirements and the viewpoint or role activated, too. In our approach, we aim to exploit economies of scale by the use of Multi-tenancy concept as the most of approaches cited as related work do. But, we also rely on the use of Multiview

 ISBN: 978-1-61208-367-4

notion predicating on the RVC components to exploit more and more economies of scale.

As future works, we will define an new artifact based on Rich-Variant Component enabling customers to choose to share or not to share with other customers. The next step will be devoted to the implementation of our approach by applying it to the case study consisting of SaaS application for managing private school so as to show its interest and improve it by tests evaluation.

VI. CONCLUSION

The variability management, the RVC component notion and the Multi-tenancy rationalization are key enablers for the accomplishment of flexibility, reusability and exploiting economies of scale in customizable SaaS applications. For this objective, we have initiated in this paper our approach which is primarily based on integrating two types of variability to create a more flexible and reusable SaaS environment while exploiting economies of scale. For this purpose, we introduced the background knowledge of our work: variability management mechanisms, Cloud Computing and Multi-tenancy. Then, we showed the need of managing variability for Cloud environments. Finally, we presented the Multiview component concept to introduce our contribution. Our present work is devoted to the implementation of our approach by applying it to a case study showing its interest.

REFERENCES

[1] R. Mietzner, "A method and implementation to Define and Provision Variable Composite Applications, and its Usage in Cloud Computing," Dissertation, University of Stuttgart, August 2010.

[2] S. Walraven, D. V. Landuyt, E. Truyen, K. Handekyn, and W. Joosen, "Efficient customization of multi-tenant Software-as-a-Service applications with service lines," Journal of Systems and Software vol. 91, Jan. 2014, pp. 48-62.

[3] S. T. Ruehl, H. Wache, and S. A. W. Verclas, "Capturing Customers' Requirements towards Mixed-Tenancy Deployments of SaaS-Applications," IEEE CLOUD, 2013, pp. 462-469.

[4] M. Aiello, P. Bulanov, and H. Groefsema, "Requirements and tools for variability management," Proc. the 2010 IEEE 34th Annual Computer Software and Applications Conference Workshops (COMPSACW '10), Washington, DC, USA, 2010, pp. 245-250, doi:10.1109/COMPSACW.2010.50.

[5] M. A. Iqbal, A. M. Zaidi, and S. Murtaza, "A new requirement prioritization model for market driven products using analytical hierarchical process," Proc. DSDE'10, IEEE, Feb. 2010, pp. 142-149.

[6] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Technical report, CMU/SEI TR-21, USA, Nov. 1990.

[7] K. C. Kang, S. Kim, J. Lee, K. Kim, G. J. Kim, and E. Shin, "FORM: A feature-oriented reuse method with domain-specific reference architectures," Annals of Software Engineering, vol. 5, 1998, pp. 143–168.

[8] M. Erwig and E. Walkingshaw, "Variation programming with the choice calculus," Generative and Transformational Techniques in Software Engineering, Springer-Verlag Berlin Heidelberg, 2012, pp. 55-100, doi:10.1007/978-3-642-35992-7_2.

[9] I. Trummer, "Cost-Optimal Provisioning of Cloud Applications," Diploma thesis, University of Stuttgart, Faculty of computer sciene, Feb. 2010.

[10] NIST. Definitoon of Cloud Computing - National Institute of Standards and Technology, Gaithersburg, MD, 2009.

[11] C. Fehling and R. Mietzner, "Composite as a Service: Cloud Application Structures, Provisioning, and Management," it - Information Technology Special Issue: Cloud Computing, April 2011, pp. 188-194.

[12] M. Zaremba, T. Vitvar, S. Bhiri, W. Derguech, and F. Gao, "Service Offer Descriptions and Expressive Search Requests - Key Enablers of Late Service Binding," Proc. 13th International Conference on E-Commerce and Web Technologies (EC-Web), Vienna, Austria, Sept. 2012, pp. 50-62, doi: 10.1007/978-3-642-32273-0_5.

[13] M. Zaremba, S. Bhiri, T. Vitvar, and M. Hauswirth, "Matchmaking of IaaS cloud computing offers leveraging linked data," Proc. *28th Annual ACM Symposium on Applied Computing* (SAC) , New York, NY, USA, 2013, pp. 383-388, doi:10.1145/2480362.2480440.

[14] B. El Asri, "A model of multiview components for VUML," National dissertation, Engineering School of Information Technology and System Analysis (ENSIAS), Rabat, Oct. 2005.

[15] M. Nassar, " VUML: a viewpoint oriented UML extension," Proc. 18th IEEE International Conference on Automated Software Engineering, Oct. 2003, pp. 373-376, doi: 10.1109/ASE.2003.1240341.

 ISBN: 978-1-61208-367-4

Applications Architecture for a Medium Sized Manufacturing Firm

Alicia Valdez
Research Center
University Autonomous of Coahuila
Coahuila, Mexico
aliciavaldez@uadec.edu.mx

Sergio Castaneda, Laura Vazquez, Azucena García
Research Center
University Autonomous of Coahuila
sergiocastaneda@uadec.edu.mx,
lauravazquez@uadec.edu.mx,
azucenagarcia@uadec.edu.mx

***Abstract*— Small and Medium Enterprises require new technologies and methods of work organization that allow them to improve their productive and competitive capacities. The Enterprise Architecture is a methodology that defines architectures for the use of the information in support of the business strategy, looking for strategic alignment between information technology and business processes. The Applications Architecture is a partial architecture of the enterprise architecture, which aims to define the best kinds of applications needed for data management and support the business processes, considering the strategic use of information and technology for the competitive advantage of the company. This research project designed and implemented an application architecture in a medium-sized manufacturing company using open source software, resulting in the identification of strategic areas of opportunity for this architecture, and the development of a basic web page to start e-commerce activities, achieving 7% increase in sales of the company, thus helping to raise productivity and competitiveness.**

Keywords-Application architecture; SME; Enterprise architecture; e-commerce

I. INTRODUCTION

Productivity and competitiveness of the Small and Medium Enterprises (SME's), are important, because they provide a high rate of employment, the Secretary of Economy estimates that 7 of each 10 employees work in SME's [1].

Therefore, it is necessary to create a strategic solution to improve the capabilities of these companies and respond quickly to the challenges, either business related or technological, which is today's markets demand [2].

Enterprise Architecture (EA) is a methodology to provide companies with a framework for the use of information on business processes in ways that support the business strategy [3].

The framework was created to provide a disciplined approach to managing information systems, and professional communication that would allow the improvement and integration of tools and development methodologies [4].

Recently the framework was updated by Ross, Weill, and Robertson [5], cited by Bijata and Piotrkowski, where the concept of Enterprise Architecture as Strategy (EAS), has been proposed as an enterprise framework, composed of three elements, operating model, enterprise architecture, and information technology cooperation model; established as an improvement of enterprise architecture, for an adjustment in the strategy of the organization [6].

Harrel and Sage have mentioned that the key to developing Enterprise Architecture is located in:

- Business processes.
- Data for processes.
- Technology.
- Interfaces with customers.
- Applications software.

where each factor has its own architecture and develops its own tools that support it [7].

In Singapore, a research project of strategic alignment between the business model and Information and Communication Technology (ICT) was developed, in the form of an architecture for small and medium engineering and construction enterprises, focused on four shafts, business strategy, strategy of ICT, business or organizational infrastructure, and infrastructure of ICT [8].

This project was established, looking for the transformation of the business sector aforementioned, to prepare them to face the challenges of the XXI century, supported by ICT, designed to acquiring new skills, using a framework of enterprise architecture, the results of the study were [8]:

- Some companies do not consider ICT as strategic to their business.
- The practice of using ICT is mainly in administrative functions.
- The exploitation of ICT for improving the technical areas is low.
- Insufficient professionalism.
- Shortage of professional and technical personnel.
- Insufficient use of technology.
- Methods of unproductive operation.

 ISBN: 978-1-61208-367-4

- Involved in domestic markets.

Ahlemann et al. [9] have proposed the EAM as a way to deal with organizational complexity and change, make it strategic for business management.

These studies have shown that results of strategic alignment between ICT and business strategy are similar in other countries.

One important activity of the pre-construction of the Applications Architecture (AA) was the analysis of business model, and the strategic planning, which discusses, among other things, the mission, vision, geography, competitive advantage, customers, suppliers, enterprise services, and other important factors that relate to the definitions of the business.

The AA is a partial architecture of the Enterprise Architecture, that aims to define the best kind of data needed to manage and support business processes applications, also known as conceptual applications model [10].

In the AA, it identifies every possible application to manage the data and support the business, considering the strategic use of ICT for competitive business advantage. As an increasing number of functions and processes within companies, it has also increased the number of computer-based information systems, which are improving the efficiency and quality of the areas and processes that support [10].

The AA is a definition of what applications will manage data, and provide information to people running business processes. Applications enable the Information Systems (IS) function to achieve its mission; this is to provide access to the necessary data in a useful format at an acceptable cost.

Using as a source of information, the definitions of data architecture, the data-matrix functions, the business model, and the applications list, to propose candidate applications.

In this case study, an applications architecture was designed, like a partial architecture of an enterprise architecture, considering key processes in manufacturing SMEs, the best practices and business modeling tools that use these companies to develop it; with the objective of supporting them in increased productivity and competitiveness. An improvement proposal was designed and implemented [11].

The methodology for the case study is shown in Figure 1, where the requirements analysis began with the description of the current applications.

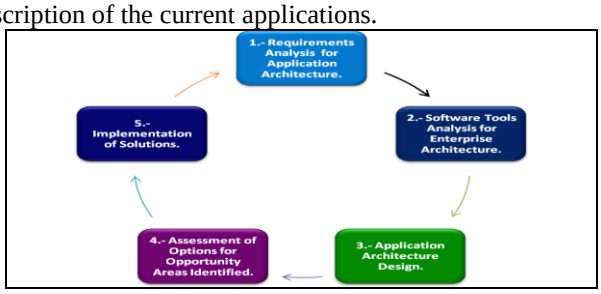


Figure 1. The methodology for AA.

The next step was a search of software tools for EA, looking for accessible tools for SME's. The selected software was Essential Architecture Manager 3.0 [12], the Figure 2 displays one view representing application capability summary, which is searching for suitable software solutions to SME's.

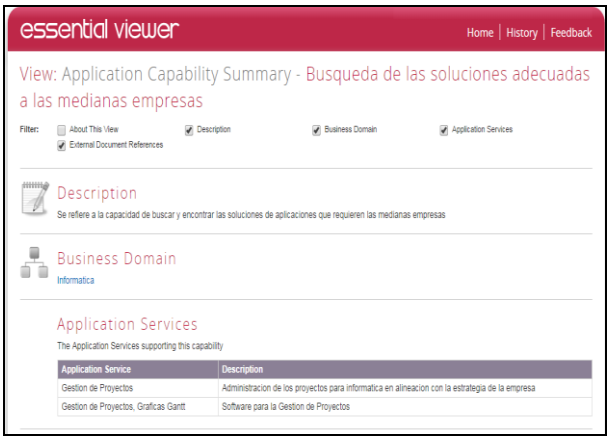


Figure 2. View of Essential Architecture Manager for AA.

The AA was designed (the next section describes the AA design widely), the assessment of options for opportunity areas identified, and finally the implementation of the solution.

The organization of this paper is as follows; Section 1 is mentioning the concepts, studies about applications architecture, and the methodology used in the case study; Section 2 describes the Application Portfolio Management (APM), as well as, the information required for the design of the application architecture from the company; finally, the implementation of the solution derived from the analysis is shown.

A. State of the Art

Op't et al. [13] have identified enterprise architecture as a key driver for governing the changes in companies, ensuring compliance in the implementation. Bernard [14] has defined EA as a holistic management, planning, and documentation activity, and has introduced the EA Cube Framework and implementation methodology. Where were defined lines of business as five sub-architectures, and three common thread areas.

- Strategic initiatives.
- Business services.
- Information flows.
- Systems and applications.
- Technology infrastructure.

The three threads are:

- Security.
- Standards.
- Workforce.

 ISBN: 978-1-61208-367-4

"Newer approaches as business services, exemplifies how EA can link strategy, business, and technology components across the enterprise within a service bus that encompasses platform independent horizontal and vertical EA components" [14].

Some frameworks were updated for including EA as a requirement, Control Objectives for Information and Related Technologies (COBIT EA), repositioned as a business framework for the governance and management of enterprise IT, defining EA such a requirement; the domains for EA in COBIT, namely business, information, data, applications, and technology [15].

EA involves key elements as, strategy, business, and technology; considering the basic Zachman's Framework [3], and notions from Spewak [9], and the new proposals as EA Cube Framework [14].

The applications architecture proposed in this paper has considered, analysis of the current applications, identifying priority processes, developing and assessment of solutions for achievement support on strategy business and technology, which are related with the general assumptions of EA, in applications architecture level.

Figure 3 shows the proposal for applications architecture.

Agility and effectiveness in operations of the manufacturing processes, as also, data sharing across the company, were some of the advantages of using this proposal.

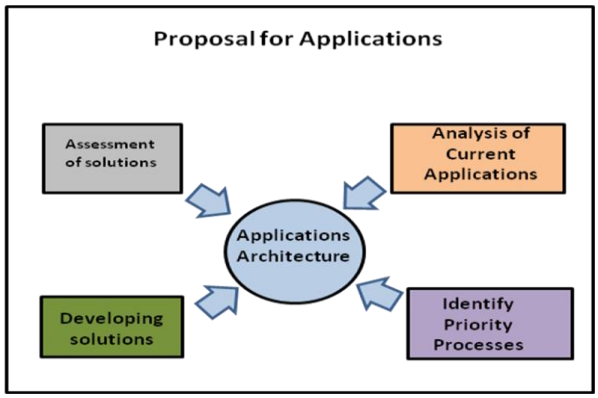


Figure 3. Proposal for Applications.

II. NOTIONS

Some researchers have identified 5 fields or categorizations related to decision making in IT [16]:

- IT Principles.
- IT Architectures.
- IT Infrastructure.
- Business needs and applications.
- IT Priorities and investments.

This set of activities have been named: The Application Portfolio Management (APM), shown in Figure 4, with the tools for each categorization [17].

Categories:

- IT Strategy: Is defined and governed by the central IT staff, who report directly to the CIO (Chief Information Office) or executive office information, considering strategic maps, strategy business, Scorecard for ICT, and Key Performance Indicators (KPI's).
- Business application needs: These needs are analyzed according to the business requirements, using Business Processes Modeling (BPM), diagrams of the company and Use Cases.

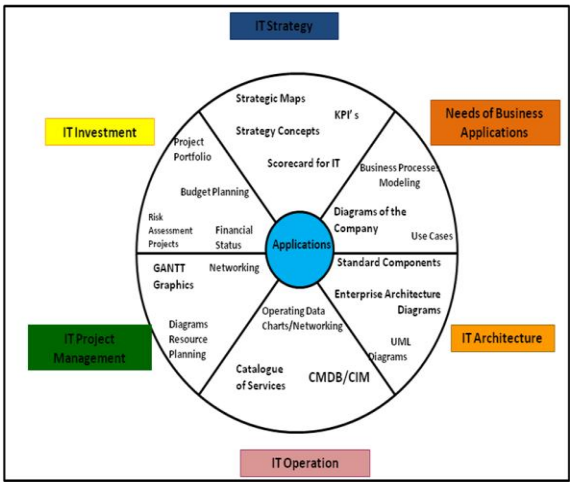


Figure 4. The Application Portfolio Management [17].

- IT Architecture: Is developed and maintained by the staff, using standard components, Enterprise Architecture diagrams, and UML diagrams.
- IT Operation: The operation is managed by the teams responsible for data centers and networks; staff also supports users in daily work, considering some tools like: Operating data charts, configuration management databases (CMDB), and catalogue of services.
- IT Project Management: Designed by staff dedicated to IT projects according to priorities, and using networking, Gantt graphics, and diagrams resource planning.
- IT Investment: Planned, negotiated and controlled by the central IT staff by cycles of annual budgets, considering project portfolio, budget planning, financial status, and risk assessment projects.

The IT staff provides support in all categories related to APM.

 ISBN: 978-1-61208-367-4

The AA is the conceptual model of business applications, composed of software applications to support business processes; the basics premises for construction of AA are shown in Table I, where the objective has been defined as the best kinds of applications to manage data and support the business processes, by using Essential EAM, as a repository for the instances obtained [18].

The technological domains identified by the EAM for applications were: systems implementation, environment services, integration services, business systems, business support services, and systems management. Each one has a set of capabilities executed by applications. In Figure 5 the domains and capabilities are shown.

TABLE I. OBJECTIVE, PRINCIPLES AND CAPABILITIES FOR AA.

AA	Name	Description
Objective	Define the best kinds of applications to manage data and support business processes.	Define the best applications that support the business processes.
Principle	Customizing minimum packaged applications.	Minimize app package, customization will improve the ability to ensure ongoing maintenance and maximum value obtained from the adoption of a package solution.
Capabilities	Analysis, design, programming and implementation of information systems. Search packaged solutions tailored to the needs of the SME's Provide technical support for software and hardware throughout the company.	Domain in the analysis, design, programming and implementation of information systems. Domain in search packaged solutions tailored to the needs of the SME's Domain to provide technical support for software and hardware.

This set of capabilities represents the broad domain for the applications in the companies. The IT staff must consider the capabilities required for the optimum functionality of the applications.

A. Information of the Applications Architecture

The first step was to collect the information about the current applications of the case study firm, for this purpose was applied a description format including the next data.

- System name.
- Project manager.
- User department.
- User contact.
- Description.
- Status.
- Long-range issues.
- Business processes supported.

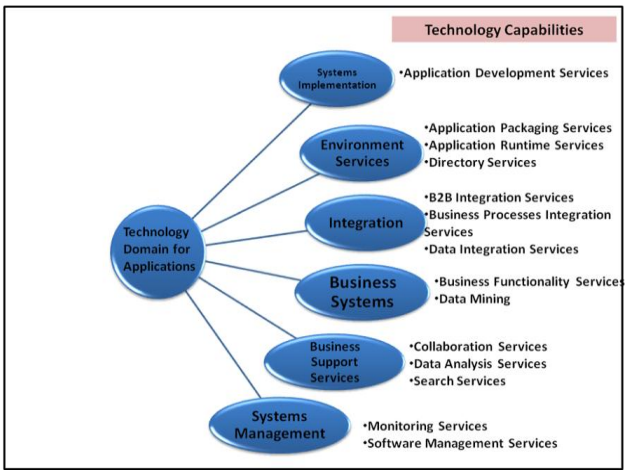


Figure 5. Technology domain for applications.

Technical aspects of the applications:

- The equipment, hardware, or physical technology platforms used.
- The networks or communication platform used.
- The software platform used.
- Preceding systems (systems that must execute before the application).
- Succeeding systems (systems that can be executed after this application has been run).

The format was applied for each application in the firm.

Subsequently, the activity for to relate the applications with the business processes of the company, was developed.

Opportunity areas for application and improvement were detected.

Table II, shows some data for the matrix of business processes-applications.

Other documents that have been analyzed for AA were:

- Data diagrams [19].
- Business model [10].

B. Design of the Application Architecture

The design of the AA has included some components that describe the context for applications, which are:

- Applications information.
- Applications function.
- Applications supported by business process
- Executed by people roles
- Relation to business process.

Figure 5 displays the components of the AA.

 ISBN: 978-1-61208-367-4

TABLE II: INSTANCES OF APPLICATION ARCHITECTURE.

Name	Description	Domain of App	Performed by business processes
Stock Information System	Management of the inputs and outputs of the company general store.	Update catalog of items, articles inventory processing.	Registry inputs and outputs of goods and raw materials.
Quality	Spreadsheets records quality of finished products.	Data of finished products according to production plan.	Verify the manufacturing process according espefications with production.
Client IS	Manage Client Portfolio.	Update Clients Portfolio,Electronic Billing.	Client Portfolio.
Financial IS	General Financial System.	Update chart of accounts, sub and sub-sub. Update Cost-Centers.policies for debit and credit accounts. Update the information of credit banks.	Accounting Manager.

The basis for the objective, principle, and capabilities for AA were previously shown in Table I.

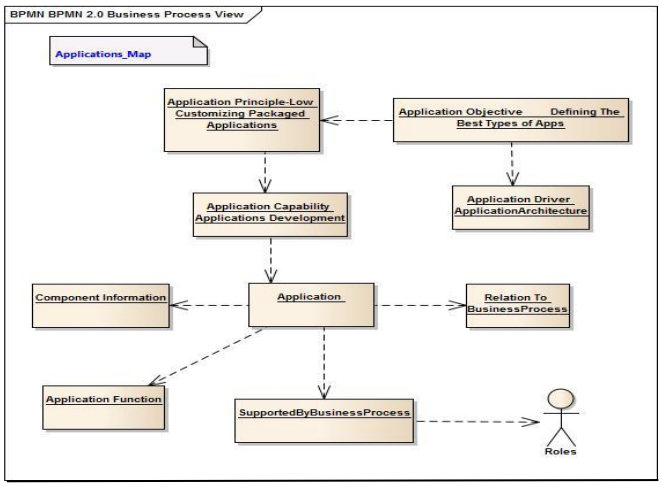


Figure 5. Application Architecture Design.

C. Needs Identified

The company has the applications support for the management processes with IS in: Distribution, Finances, Clients, Sales and Marketing, Product development, Stock and others management processes, which are shown in Table III.

The production and quality processes are supported by spreadsheets, as these processes are essential in this industry, has been programmed the acquisition, in short term, of software to streamline processes throughout the logistics chain.

By the other hand the electronic commerce is null, then the recommendation is begin with a basic website that including information about:

- Background of the company.
- Products and services.
- Scheme of manufacture.
- Quality model.
- Clients.
- Contact.
- Important information about the company.

The website was implemented by September 2013; a rise in the sells was of 7% during the next two months.

The graphic with the access statistics website is shown in Figure 6.

We are stressing–out that the company did not have a website before, only mentioned in some industrial directories as “Infomaquila”; when entering the e-commerce in this first phase, there has been an increase of the hits and visits to the website, resulting in a rise of the customers and sales.

TABLE III: APPLICATIONS OF THE COMPANY.

Company area	Application	Activities
Distribution	Shipping Information System.	
Finance	Finances Information Systems.	
Human Resources	Personnel administration.	Detect training needs of business areas, especially productive areas for develop entrepreneurial training program.
Investment Administration	Investments of the company.	
IT	Provision of IT support for company's business processes.	
Quality	Manufacture that meets production specifications.	Testing and inspection using ultrasonic methods or industrial inspection.
Sales and Marketing	Management customers. Customer service.	Continuous communication with customers to identify needs and complaints.
Stock	Register the inputs and outputs of goods and raw materials.	Suppliers management.
Product development	Program production cycles.	Cutting, marking, machining and forming of steel plates and profiles.

 ISBN: 978-1-61208-367-4

The improvement proposal will be gradually implemented, starting with applications and technology that directly impacts on priority processes as quality and production; continuing with human resources processes, the Table IV shown estimated resources for opportunity areas.

TABLE IV: ESTIMATED RESOURCES FOR OPPORTUNITY AREAS.

Application	Estimated price	Delivery time	Requirements for installation
Manufacturing System	$ 3,000.00	60 to 80 days.	Windows Server 2003 or higher, 2 Ghz or higher processor, 4Gb Ram, SQL Server Express Edition.
Information Systems for Management Training.	$ 1,000.00	21 to 30 days.	Windows 7 or higher.
Proposal to integrate e-commerce	$ 1,000.00	21 to 30 days.	Windows Server.

Approximate calculations to June, 2014 in USD.

The manufacturing system has been the most important need detected, since it would share information with production and quality, furthermore shipment and finances.

Figure 6. Access statistics website

The next stage of the website is to add popular search engines like Google, Bing, and, Yahoo; links and social networking accounts, Google maps location, bilingual website, online payments through Paypal system, and others internet characteristics.

III. CONCLUSION

The final recommendations are for using software, hardware, and next generation networks, with efforts of successful practices for manufacturing industry that would support key processes, and help incorporate them into international markets, always looking for a return on efficient investment.

This project helped to meet the needs of SME's companies to propose affordable solutions that make business management resources and technology to solve problems.

The contribution of the paper focuses on the improvement proposal for the case study firm and the development of applications solutions, detected by the analysis.

The AA takes components of the Business Architecture, and is associated with the Technology Architecture to produce the EA complete.

Other findings in terms of improvement were: SME's have demonstrated alignment with business strategy to drive a strong organizational culture and technological infrastructure.

The company has acquired new skills through ICT.

The sharing of information with customers and suppliers has improved considerably with the use of e-commerce and networking.

REFERENCES

[1] Secretary of Economy, "SMEs news",http://economia.gob.mx/ [retrieved: 04-2014].

[2] G. Lopez and H. Tan, "Impact evaluation of SME programs in Latin America and the Caribbean", World Bank, Washington, USA, pp. 4-10, 2010.

[3] J. Zachman, "A framework for information systems architecture", IBM systems journal, vol 26, 1987, pp. 276-292.

[4] J. Zachman, "Enterprise architecture artifacts vs. application development artifacts (Part 2)", http://www.itu.dk [retrieved: 05-2014].

[5] J. Ross, P. Weill, and D. Robertson, "Enterprise Architecture as Strategy" , Harvard Business School Press, USA, 2006, pp. 10-20.

[6] M. Bijata and K. Piotrkowski, "Enterprise architecture as a tool to support the strategic management process in an organization", Hyperion International Journal of Econophysics & New Economy, vol. 7, pp. 177-189, 06- 2014.

[7] M. Harrel and A. Sage, "An enterprise architecture methodology to address the Enterprise Dilemma", Journal of Information Knowledge Systems Management, vol. 9, 2010, pp. 211-237.

[8] B. Goh, "Applying the strategic alignment model to business and ICT strategies of Singapore's small and medium sized architecture, engineering and construction enterprises", Journal of Construction Management and Economics, vol. 25, 02-2007, pp. 157-169.

[9] F. Ahlemann, E. Stettiner, M. Messerschmidt, and C. Legner, "Strategic Enterprise Architecture Management" , Springer, Germany, 2013, pp. 5-16.

 ISBN: 978-1-61208-367-4

[10] S. Spewak and S. Hill, ”Enterprise architecture planning, developing a blueprint for data, application and technology”, Wiley publisher, USA, 1992, pp. 1-6.

[11] J. Schekkerman, “Enterprise architecture good practices guide: How to manage the enterprise architecture practice”, Trafford publisher, 2008, pp. 15-20.

[12] D. Rice, “Review of essential architecture manager 1.0”, Journal of enterprise architecture, vol. 1, 05-2009, pp. 1-7.

[13] M. Op’t, E. Proper, M. Waage, J. Cloo, and C. Steghuis, “Enterprise architecture creating value by informed governance”, Springer, Netherlands, 2009, pp. 24-31.

[14] S. Bernard, “Enterprise architecture linking strategy, business, and technology”, AutorHouse, Third edition, USA, 2012, pp. 25-31.

[15] COBIT, ”COBIT 5 makes enterprise architecture a mandatory discipline”,http://companies.mybroadband.co.za/ [retrieved: 07-2014].

[16] P. Weill and J. Ross, “IT governance how top performers manage IT decisions rights for superior results”, Harvard Business School, Boston, USA, vol. 320, 2004.

[17] G. Riempp and S. Gieffers-Ankel, “Application portfolio management: a decision-oriented view of enterprise architecture”, Journal of Information Systems & e-Business Management, vol. 5, 08-2007, pp. 359-378.

[18] The Essential project,”The Essential Project”, 2013, http://www.enterprise-architecture.org/ [retrieved: 03-2014].

[19] R. Elmasri and S. Navathe, “Fundamentals of database systems”, Addison-Wesley, Sixth edition, USA, 2010.

 ISBN: 978-1-61208-367-4

Enhanced Search: An Approach to the Maintenance of Services Oriented Architectures

Norman Wilde, Douglas Leal, George Goehring, Christopher Terry
Department of Computer Science
University of West Florida
Pensacola, FL, USA
e-mail: nwilde@uwf.edu, douglas.leal@gmail.com, pensacoder@gmail.com, cterry@students.uwf.edu

Abstract— **This paper describes the use of search techniques to ease the burden of software maintenance for Services Oriented Architecture composite applications. Services Oriented Architecture is a paradigm that offers many potential business and social benefits, especially because it creates opportunities for composite software applications that share data and functionality across organizational boundaries. However, along with these benefits will come new challenges in the maintenance of these applications. The first necessity in any software maintenance task is to comprehend how the existing software functions. To gain this comprehension, maintainers will need to study a bewildering variety of artifacts, ranging from XML-based interface descriptions, through source code in a variety of languages, to traditional text documents in many different formats. For some years, we have been experimenting with the use of modern search techniques, enhanced where possible by rule-based reasoning, to aid maintainers of composite applications in gathering the information they will need to do their jobs. In this paper, we describe version 2 of our SOAMiner search system and discuss how its design emerged from our experiences. While SOAMiner is still a prototype, we argue that search, enhanced and specialized for Services Oriented Architecture can provide useful support to maintainers of these very heterogeneous applications.**

Keywords-Services Oriented Architecture; SOA; Software Maintenance; Search; Rule-Based Systems.

I. INTRODUCTION

The last decade has seen the emergence of a new paradigm for large scale software applications often called Services Oriented Architecture (SOA). While definitions of SOA vary, the term usually refers to large *composite applications* implemented as large-grained services running on different nodes and communicating by message passing (see Figure 1). Implementation technologies differ, but often follow the Web Services interoperability standards.

The SOA architectural style has great potential to achieve business or social goals through interoperability across organizational boundaries. As an example of SOA, consider the CONNECT project, which provides a set of software and standard interfaces for health information exchanges in the United States [1]. The goal of CONNECT is to enable health data to follow a patient wherever he may need treatment.

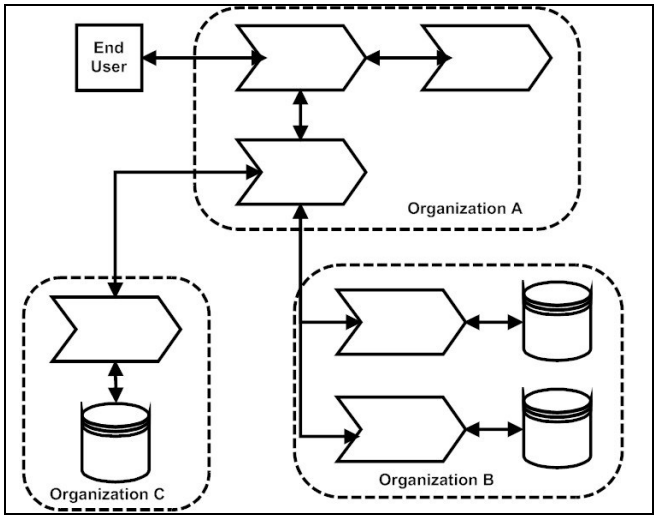


Figure 1. A SOA composite application with services from three partner organizations exchanging messages.

However, to achieve such benefits over the long term, SOA composite applications will have to be maintainable in a rapidly changing world. Several authors have pointed out characteristics of SOA that may make maintenance difficult [2][3][4]. Often, one such characteristic is distributed ownership, so that different services in the composite application are operated and maintained by different partner organizations. Thus, changes to specifications may need to be negotiated, coordination of updates may be complicated and the maintainer's information about some services may be incomplete. The mix of partners may change unpredictably over the application's lifetime, requiring quick re-engineering to adapt as services are offered or withdrawn. Critical security issues may emerge without warning, and it may be difficult to identify their impacts without knowing how partner services are implemented.

A traditional stumbling block in all software maintenance has been the need for program comprehension. The first question a maintainer must always ask is "how does the software work now?" Changes made to a software system without deep understanding can be highly error prone. A particular maintainer's problem has always been the *delocalized software plan* in which the original programmer's

 ISBN: 978-1-61208-367-4

strategy for addressing some specific issue has been implemented by related code in several distant program modules [5]. Subtle faults may be introduced if a maintainer makes changes in one of these modules in ignorance of possible effects in others.

In SOA, the delocalization is not confined to a single executable but may spread across different services which, as we have seen, may have different owners. While every service has a published interface, which is sufficient to invoke it, in practice there are often additional data and operation sequencing constraints that must be learned by experience or by study of documentation [6].

There has been a modest amount of recent research on maintaining SOA applications. Papazoglou, Andrikopoulos, and Benbernou categorize changes into "deep" and "shallow" and discuss how to keep services compatible [7]. Several authors have proposed dynamic analysis approaches that analyze inter-process messages to pull together a view of execution across the multiple services. An early tool of this kind was IBM's Web Services Navigator, which provides several visualizations of message logs [8]. A later paper from the same group describes a process that looks more deeply into message contents to identify data correlations between different messages [9]. Yousefi and Sartipi propose analyzing dynamic call trees from distributed execution traces to identify features in a SOA application [10]. A different reverse engineering approach, which does not rely on executing the system, recovers concept maps from the interface descriptions as a starting point for knowledge engineering interviews with system experts [11].

Looking for a simpler and more flexible approach, for some time our group has been researching ways to exploit the power of modern search techniques and adapt them to the specific needs of SOA maintainers. The overall project is called SOAMiner and has gone through a series of prototyping and exploration phases [12]. In this paper, we will describe version 2.0 of SOAMiner, which incorporates the experience from these earlier studies. SOAMiner is built on top of the Apache Solr™ open-source search platform [13]. The new version of SOAMiner provides a combination of conventional text search, specialized search that exploits the structure of many SOA artifacts, and rule-base abstraction to provide summarized descriptions of SOA services and data.

In the next section of this paper, we explain how these three strategies emerged from our experience in applying search to SOA. Then, in Section III, we illustrate their application by showing how SOAMiner can address a maintenance scenario for a simple SOA composite application. Finally, in Section IV, we conclude with some thoughts about SOA and the evolution of SOA systems.

II. SEARCHING SOA ARTIFACTS

In trying to comprehend a SOA composite application, a maintainer must deal with a bewildering variety of artifacts. These may include XML documents that describe service interfaces, source code for service implementations, and any conventional documentation that a service provider has chosen to offer. In developing a search strategy for these different classes of artifacts a key decision is the granularity of response. If a search returns just the few words that match the query, then the maintainer will struggle to understand how these fit into the application as a whole. If a large volume of surrounding text is also returned, then the maintainer may be buried in extraneous details. In this section, we discuss our experiences in searching these different classes of SOA artifacts and the granularity we have chosen for each class.

A. Searching XML Artifacts

When SOA is implemented using Web Services, then much of the information about each service is coded in XML format as specified in one or more of the Web Services Standards ([14], Chapter 16). The most common standards cover Web Services Description Language (WSDL) to specify how to call a service and XML Schema Definitions (XSD) to specify the data exchanged in messages. Some SOA systems also use Business Process Execution Language (BPEL) which is essentially a programming language encoded in XML for orchestrating interactions among services.

The XML artifacts may often be very large; we have seen extreme WSDL's of over 1 MB and several thousand lines is not uncommon for an XSD. Such files are often generated by some tool but it may still be necessary for the maintainer to study them himself when trying to comprehend a service. The structure of these files does not facilitate human navigation.

For example, to identify the data types being used by a particular service a maintainer needs to read its WSDL "bottom up", starting from a <service> tag near the end, locating the <port> tag it contains, navigating from there to a referenced <binding> tag, which in turn references the <portType>. From there the <portType> encloses a set of <operations> with <input> and <output> tags each pointing to a <message> tag. However, the maintainer is still not finished because in most cases each <message> simply references the actual data types, which are either enclosed within the <types> section near the beginning of the WSDL, or possibly contained within a completely separate XSD file [14].

Generic search approaches, such as a text editor's 'find' or a document-oriented web search engine, do not work very well on these XML artifacts. Such approaches ignore too much context because they are unaware of the significance of XML tag names and of the information conveyed by element nesting. Figure 2 provides one example showing how a port type is defined in a WSDL. Element nesting determines that the messages relate to the operation and the operation to the port type.

```
<!-- portType for  the InventoryRepository process -->
<portType name="InventoryRepositoryPortType">
  <operation name="checkInventory">
    <input  message="tns:InventoryRepositoryRequestMessage" />
    <output message="tns:InventoryRepositoryResponseMessage"/>
  </operation>
</portType>
```

Figure 2. Portion of a WSDL showing the definition of an operation within a port type.

When SOAMiner searches XML, the basic granularity is the element start tag, so that a search for "checkInventory" would return just the <operation> tag from Figure 2. If the system is large, the user can specify a faceted search to limit the results to a single tag type. As well, in SOAMiner we also attach to each tag its parent and any children in the XML document. Thus, if using our search GUI, the user could hover over that result and see the surrounding <portType> and the <input> and <output> tags. That provides the maintainer with a few more hints as to the context of each search result so he can focus quickly on the results that are of most interest.

B. *Rule-Based Abstraction from XML Artifacts*

However, we can do even better than that by exploiting knowledge about the semantics of the different XML tags through a process of rule-based abstraction. We have implemented such abstractions in a component of SOAMiner called SOAIntel. An expert can specify a set of rules for SOAIntel to define an abstraction, which summarizes some characteristic of a class of SOA implementations. For example, the rules could describe the above mentioned chain of reasoning to relate the service to the data items in its input and output messages. The resulting abstraction would be a compact description of the service, its operations, and their messages.

Rules are encoded using the DROOLS Expert rule-based system [15]. SOAIntel uses the rules and the DROOLS reasoning engine to analyze the XML inputs and produce a set of abstractions. These abstractions are then loaded into the SOAMiner index so that they may also be returned by SOAMiner searches. Thus, a maintainer searching on "checkInventory" would also find that this operation is part of a service abstraction named InventoryRepository and thus see that its messages use an element called inventoryQuery, etc.

The rule-based abstraction process is very flexible, so that new rule sets can easily be added to cope with changes to the Web Services standards or with specific maintenance needs for any particular class of composite applications [16].

C. *Searching Source Code and Documentation*

The source code for a SOA service may be in any of a multitude of languages; in fact one of the objectives of SOA is to allow services written in one language to invoke transparently services written in another. In several of the most common languages, such as Java and C#, much of the code is commonly generated within an Integrated Development Environment (IDE). For example, a Java developer using NetBeans will call a tool called *wsimport* to read a WSDL interface description and generate Java classes for the message data types and a shell service implementation. The generated code can be rather obscure, and as well makes use of many Java annotations to guide the run-time environment as the service executes. While the availability of generated code greatly reduces the amount of code a service developer needs to program, it also creates complex mechanisms that a maintainer may need to learn.

The diversity of source languages and run-time mechanisms makes it very difficult to develop a general code search tool with any intelligence. Instead, for now, SOAMiner falls back on normal text search, to locate lines of code matching a given query string.

The situation is similar for natural language documentation, which may be in text, Portable Document Format (PDF), HyperText Markup Language (HTML) or some word processor format. In the future, it may be possible to apply text mining techniques such as text classification and text clustering to these documents, but for now SOAMiner relies on general text search, using the facilities of Apache Tika™ to parse each document format and extract the text contents [17]. Since lines and even paragraphing may not be meaningful for all document types, each SOAMiner search simply returns the entire document contents.

D. *Search semantics for SOA*

Software Engineers search software for many reasons, but two very common ones are *concept location* [18] and *impact analysis* [19]. Concept location has to do with finding the places in a software system where some particular concept is addressed. For example, one could ask "where are font changes handled in this word processor?" and search for the concept "font" in code, documentation, etc. On the other hand, impact analysis is concerned with establishing the scope of a needed change. If, for example, the Software Engineer has determined that a particular function needs to be modified, then he needs to look at all the places that function is called so that he can see what the change may impact.

One of the observations we made after working with the first versions of SOAMiner was that the semantics of these two kinds of search are really quite different. Concept location will usually use natural language semantics and most of the techniques used in search engines should be applied. For example, queries should be stemmed and case insensitive, so that "font" will match "fonts" or "Font". Query words should break on punctuation or case changes so that again "font" will match "font_change" or "fontChange".

However, for impact analysis the rules should be very different and use identifier semantics. Normally the Software Engineer will have located a particular variable or function name, such as "fontChange", and only wants to locate occurrences of that identifier. A search with natural language semantics would return all text where either "font" or "change" appeared, and that would be far too many places to examine.

The solution adopted in version 2 of SOAMiner is to provide two alternate indexes, one using natural language semantics and the other using more restrictive identifier semantics. The user may choose which to use for any particular query depending on the results sought.

III. ILLUSTRATIVE EXAMPLE

To provide an example of the power of enhanced search, we may apply it to a simple SOA composite application called WebAutoParts.com, a hypothetical on-line automobile

 ISBN: 978-1-61208-367-4

parts dealer. The owners of WebAutoParts have adopted an agile development strategy, in which a small amount of internal code orchestrates commercially available services to provide needed functionality quickly [20]. WebAutoParts is an academic system, not a real application, so several of its components are stubs instead of full code. Still it models the complexity of a real application since it consists of in-house services with BPEL and some other code artifacts, WSDL artifacts that describe external services from well know vendors (e.g., Amazon Web Services, StrikeIron.com), and XSD schemas to define data types used in system messages (see Table I). The application provides an order processing work flow (see Figure 3) in which incoming orders are first checked to confirm that inventory is available, then sales tax and shipping are computed, and finally the order is stored and a note placed in a message queue to trigger order fulfillment (packing and shipping).

To illustrate the use of SOAIntel, two rule sets were written that generate two different kinds of abstractions from the XML files. The first abstraction is a compact service summary that shows the service and port type, the operations in that port type, and the names of the input and output messages of each operation. For the great majority of services this summary will obviate the need to step through the WSDL tag by tag to understand the service interface.

The second rule set generates a data type summary abstraction that shows the different data items making up a message. During our earlier studies with the first version of SOAMiner users requested this kind of summary to help them navigate the complexities of data typing in SOA [21]. The Web Services standards give developers a wide variety of ways to define the data in messages, and the definitions may look very different even if the final message content is much the same. For example, the data definitions may be in different places, either in the <types> section of the WSDL itself, or else located in an associated XSD file.

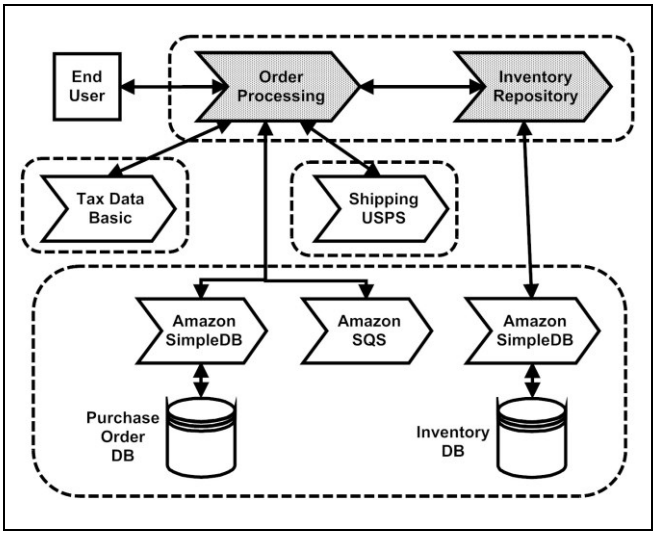


Figure 3. The WebAutoParts order processing work flow showing internal (shaded) and external services.

TABLE I. WEBAUTOPARTS ARTIFACTS

File Type	Files	Lines
WSDL (XML)	6	2433
BPEL (XML)	2	189
XSD (XML)	2	64
JAVA (Code)	6	450
C# (Code)	3	336
Microsoft Word	1	718
HTML	1	374
PDF	1	230

The style of the definition can also vary widely since developers may use different combinations of XSD elements, references and complex types to say much the same thing. (The different design patterns have been given names such as *Russian Doll* and *Venetian Blind*, and each has its own advantages and drawbacks in terms of re-usability and visibility [22]). To reduce this confusion our second rule set extracts a simple list of the data items making up each message, independent of the location or form of the definition.

To see how a maintainer could use enhanced search in studying an application such as WebAutoParts, consider the following hypothetical scenario. Employees of WebAutoParts have reported that, occasionally when packing and shipping an order, an item is found to be out of stock, even though the order processing workflow showed that inventory was available. Something in the computation of stock levels is obviously in error. The problem is passed to a software engineer for action. Let us suppose that this software engineer has little previous experience with the order processing work flow of Figure 3.

Table I enumerates the artifacts that describe WebAutoParts. There are a total of 10 XML files, 9 code files and 3 documentation files. These are loaded into the SOAMiner Solr index using the parsers for XML, code and documentation respectively.

As always, the software engineer's first question is "How are stock levels computed now?" He uses SOAMiner to do a concept location query on "stock". The results are shown in column A of Table II. Just one documentation file was located and he picks that as the starting point most likely to give him an overview of the situation. The documentation file turns out to provide a general description of order processing and provides roughly the same information that readers of this paper have already seen. While it mentions briefly that stock levels are checked it does not say how. It does, however, show the overall workflow and indicates what services participate in it.

TABLE II. RESULTS OF QUERIES ON WEBAUTOPARTS

	Column A concept location "stock"	Column B impact analysis "numberInStock"
XML tags	2 element tags	2 element tags
Abstractions	2 message data items abstractions	2 message data items abstractions
Code lines	13 Java, 6 C#	9 Java, 3 C#
Documentation files	1 Word doc	none

 ISBN: 978-1-61208-367-4

The software engineer next looks at the two abstractions, which show the data items making up the InventoryRepositoryRequestMessage and the InventoryRepositoryResponseMessage. He can see immediately that these are respectively the input and output messages of an operation called checkInventory in the InventoryRepository service. His query on "stock" matched a data item named "numberInStock" which is contained in both messages. (Concept location queries use the natural language semantics index in which words break on changes of case, so the query word "stock" matches "numberInStock".)

It seems highly likely that the error involves in some way the numberInStock data item and the checkInventory operation. Thus next the software engineer does an impact analysis query on "numberInStock". The query uses the identifier semantics index so it will only find exact matches to that string. The results are shown in Column B of Table II. The query finds the same two XML tags and message data items abstractions, but it locates a smaller set of code lines, reducing the places the software engineer needs to look. The code lines are in a Java implementation of the InventoryRepository service and a shell C# implementation of a test client to that service.

Now that he has the big picture, the software engineer can start looking at code. Here he can make use of specialized IDE's for Java or C# having their own very good search facilities. Combining his overall view of the workflow with a little analysis reveals a classic "omitted logic" problem [23]; while InventoryRepository gets the correct value for the numberInStock at any moment, there is nothing to prevent a second order from checking that same stock level before the first has completed order fulfillment, so the same stock may be committed twice. As often occurs, the error is not really "in" any particular service, but is a consequence of implicit assumptions made as the different services were orchestrated together.

IV. CONCLUSIONS

In this paper, we have argued that Services Oriented Architectures will not attain their full potential unless these applications can be rapidly maintained. SOA applications will need to provide high availability in a world with changing requirements, shifting partner alliances and emergent security threats. Their maintainers will need to gather information quickly to comprehend and respond correctly to each challenge.

In confronting these challenges, maintainers will need both good governance and good tools. In SOA, the term "governance" refers to the set of policies, rules, and enforcement mechanisms for developing, using, and evolving SOA-based systems [24]. There is a great danger of organizations trying to go too far too fast with SOA and creating composite applications that go beyond the organization's capacity to maintain. The scope of applications, the range of implementation technologies and the rate of requirements creep need to be limited to match organizational capabilities.

If the organization provides a reasonable governance framework, then well qualified software engineers with good tools should be able to do the job. Our SOAMiner is only intended as one example of the sorts of tools that will be needed. The current version remains a prototype. There are some places where it is less precise than we would wish, for example in the handling of namespaces. The user interface remains a work in progress. However, we feel that the flexibility provided by the combination of modern search with rule-based abstraction is well suited to the changing world of SOA. The search techniques can be applied to just about any kind of artifact encountered in a SOA system, while the abstraction mechanism can leverage a rule base that grows with experience. Thus, a search tool like SOAMiner can provide some useful information almost all the time, and can provide better and better information as experience grows.

It will be interesting to see how well the SOAMiner approach scales to real-world SOA. Limited experience with one larger system indicated that the pure search aspects provided excellent performance, which was to be expected since the Solr search engine was developed with large data sets in mind. The scalability of the rule-based abstractions may be more problematic. Our limited experience so far is that performance can depend on how well the rules are crafted to exploit the DROOLS index structure.

The evolution of SOA systems will never be easy, but with thoughtful governance, skilled software engineers and good tools, it should be possible to manage the challenges.

ACKNOWLEDGMENT

Work described in this paper was partially supported by the University of West Florida Foundation under the Nystul Eminent Scholar Endowment. Apache, Apache Solr, and Apache Tika are trademarks of The Apache Software Foundation. Used with permission. No endorsement by The Apache Software Foundation is implied by the use of these marks.

REFERENCES

[1] "What is CONNECT?", Internet: http://www.connectopensource.org/about/what-is-connect, link accessed 2014.07.22.

[2] N. Gold and K. Bennett, "Program comprehension for web services", International Workshop on Program Comprehension (IWPC'04), June 2004, pp. 151-160, doi: 10.1109/wpc.2004.1311057.

[3] N. Gold, C. Knight, A. Mohan, and M. Munro, "Understanding service-oriented software", IEEE Software, Vol. 21, March 2004, pp. 71-77, doi: 10.1109/ms.2004.1270766.

[4] G. Lewis and D. Smith, "Service-Oriented Architecture and its implications for software maintenance and evolution", Frontiers of Software Maintenance, FoSM 2008, Sept. 2008, pp. 1-10, doi: 10.1109/fosm.2008.4659243.

[5] S. Letovsky and E. Soloway, "Delocalized plans and program comprehension", IEEE Software, vol.3, no.3, May 1986, pp. 41-49, doi: 10.1109/MS.1986.233414.

[6] S. Halle, T. Bultan, G. Hughes, M. Alkhalaf, and R. Villemaire, "Runtime verification of web service interface

 ISBN: 978-1-61208-367-4

contracts", IEEE Computer, Vol. 43, March 2010, pp. 59-66, doi: 10.1109/mc.2010.76.

[7] M. P. Papazoglou, V. Andrikopoulos, and S. Benbernou, "Managing Evolving Services," IEEE Software, Vol. 28, No. 3, May/June 2011, pp. 49-55, doi: 10.1109/MS.2011.26.

[8] W. De Pauw, et al., "Web services navigator: visualizing the execution of web services", IBM Systems Journal, Vol. 44, No. 4, Oct. 2005, pp. 821-845, doi: 10.1147/sj.444.0821.

[9] W. De Pauw, R. Hoch, and Y. Huang, "Discovering Conversations in Web Services Using Semantic Correlation Analysis", IEEE 20th International Conference on Web Services, ICWS'2007, July 2007, pp. 639-646, doi: 10.1109/ICWS.2007.200.

[10] A. Yousefi and K. Sartipi, "Identifying distributed features in SOA by mining dynamic call trees", IEEE International Conference on Software Maintenance (ICSM), Sept. 2011, pp. 73-82, doi: 10.1109/ICSM.2011.6080774.

[11] J. Coffey, T. Reichherzer, B. Owsnick-Klewe, and N. Wilde, "Automated Concept Map Generation from Service-Oriented Architecture Artifacts", Proc. of the Fifth Int. Conference on Concept Mapping CMC2012, Sept. 2012, pp. 49-56.

[12] E. El-Sheikh, et al., "Towards enhanced program comprehension for service oriented architecture (SOA) Systems", Journal of Software Engineering and Applications, Vol. 6, No. 9, Sept. 2013, pp. 435-445, doi: 10.4236/jsea.2013.69054.

[13] "Apache Lucine, Apache Solr", Internet: https://lucene.apache.org/solr/, link accessed 2014.07.22.

[14] N. Josuttis, SOA in Practice: The Art of Distributed System Design, O'Reilly, 2007, ISBN: 0-596-52955-4.

[15] "Drools - JBoss Community", Internet: http://drools.jboss.org/, link accessed 2014.07.22.

[16] G. Goehring, et al., "A knowledge-based system approach for extracting abstractions from service oriented architecture artifacts", International Journal of Advanced Research in Artificial Intelligence (IJARAI), Vol. 2, No.3, 2013, pp. 45-52, doi: 10.14569/IJARAI.2013.020307.

[17] "Apache Tika", Internet: http://tika.apache.org/, link accessed 2014.07.22.

[18] V. Rajlich and N. Wilde, "The role of concepts in program comprehension", 10th International Workshop on Program Comprehension, June 2002, pp. 271-278, doi: 10.1109/WPC.2002.1021348.

[19] M. Petrenko and V. Rajlich, "Variable granularity for improving precision of impact analysis", International Conference on Program Comprehension, 2009. ICPC'09, May 2009, pp. 10-19, doi: 10.1109/ICPC.2009.5090023.

[20] N. Wilde, J. Coffey, T. Reichherzer, and L. White, "Open SOALab: Case study artifacts for SOA research and education", Principles of Engineering Service-Oriented Systems, PESOS 2012, June 2012, pp. 59-60, doi: 10.1109/PESOS.2012.6225941.

[21] L. White, et al., "Understanding interoperable systems: Challenges for the maintenance of SOA applications", 45th Hawaii International Conference on System Sciences (HICSS), January 2012, pp. 2199-2206, doi: 10.1109/HICSS.2012.614.

[22] E. Hewitt, Java SOA Cookbook, O'Reilly, 2009, ISBN: 978-0-596-52072-4.

[23] R. L. Glass, "Persistent software errors", IEEE Transactions on Software Engineering, Vol. SE-2, No. 2, March 1981, pp 162-168, doi: 10.1109/TSE.1981.230831.

[24] G. Lewis and D. Smith, "Four pillars of service-oriented architecture", CrossTalk, September 2007, pp. 10-13. Available from: http://www.crosstalkonline.org/storage/issue-archives/2007/200709/200709-Lewis.pdf, link accessed 2014.07.22.

 ISBN: 978-1-61208-367-4

Evaluation of the Applicability of CM3: Emergency Problem Management within the Industry

Mira Kajko-Mattsson
ICT, KTH Royal Institute of Technology
Kista, Sweden
mekm2@kth.se

Joakim Snygg, Emil Hammargren
DSV, Stockholm University
Kista, Sweden
snygg@dsv.su.se, emil-ham@dsv.su.se

Abstract— **Software has become one of the main villains of many organizational problems, emergencies and crises. Despite this, there is only one process model defining how to manage emergency software problems. It is CM^3: Emergency Problem Management. In this paper, we evaluate the applicability of the *CM^3: Emergency Problem Management* model within five companies. Our results show that the model correctly manages the real-life emergency and crisis situations that are dependent on malfunctioning software. This evaluation shows that all the five companies have emergency processes that reflect CM^3 model's architecture, however, to different degrees. Additionally all the five companies have also designated roles that act as focal points of information and decision making during emergencies. Finally, only one company has identified the organizations and systems, which should be affected by the emergency process**

Keywords-Problem management; operational levels; task force team; software maintenance.

I. INTRODUCTION

More and more of the emergencies and crises encountered today get generated due to malfunctioning software [1][6]. Many times, their underlying software problems may be of unpredictable and uncertain dimensions [16]. Being of high severity, they may threaten to harm the organizations' businesses and survival, their stakeholders, or the general public [8][10]. For this reason, software organizations must be well prepared for protecting themselves against all types of crises and emergencies by creating a well-defined emergency and crisis management process. It is only then they may guard themselves against all kinds of unexpected financial, political, legal, media and governmental impact and consequences. [7][9][11]

Emergency problem management is recognized as an important maintenance activity type by the International Software Engineering Standard - ISO/IEC 14764 [15]. Despite this, there are no process models providing guidelines for how to manage unexpected emergency and crisis problems. To our knowledge, there is only one model dedicated to software emergencies and crises today. It is *CM^3: Emergency Problem Management* [4][5]. CM^3 stands for *Corrective Maintenance Management Model*.

TABLE I. THE FIVE COMPANIES

Company	*Nr of Employees*	*Nr of IT Employees*	*Domain*
SAS	> 12 000	≈ 150	Aviation
Northern Finance	> 15 000	> 100	Finance
Bank and Loans	> 15 000	≈ 700	Finance
Good Things	> 25 000	≈ 500	Retail
Gladstone	≈ 1 000	≈ 1 000	Gaming

CM^3: Emergency Problem Management was initially designed at *Scandinavian Airline Systems* (SAS) [4][5]. Hence, it reflected the status of SAS emergency process model. In this, paper, we study five industrial emergency processes with the purpose of evaluating *CM^3: Emergency Problem Management* and further extend it with more process elements. The five companies are SAS, *Northern Finance, Loans and Bank, Good Things and Gladstone.* Except for SAS, the companies have requested to stay anonymous. Hence, we use their fictitious names. The companies are briefly presented in Table I.

Scandinavian Airlines (SAS) is an aviation company member and cofounder of the Star Alliance. SAS is the ninth-largest airline in Europe.

Northern Finance operates within the financial sector. They are a worldwide finance company with offices from Asia to North America. However, their main business market is located in Europe. The company provides products and services in the financial sector such as trading, management and insurances.

Bank & Loan ltd. works within the financial sector and offers retail banking, asset management and financial services. They have offices in Asia, Europe, and North America, but their main business is in Scandinavia.

Good Thing Sales is one of the largest retail companies in Scandinavia with more than 1500 retail stores. Finally, *Gladstone Gamer* is one of the largest online gaming companies in the world. However, compared to the other companies in the study, this company is the youngest. Most of the employees are concerned with different aspects of IT.

The remainder of this paper is as follows. Section II presents our research method. Section III presents the extended version of *CM^3: Emergency Problem Management* Section IV describes how it matches the industrial emergency processes, and finally, Section V makes conclusions and suggestions for future work.

 ISBN: 978-1-61208-367-4

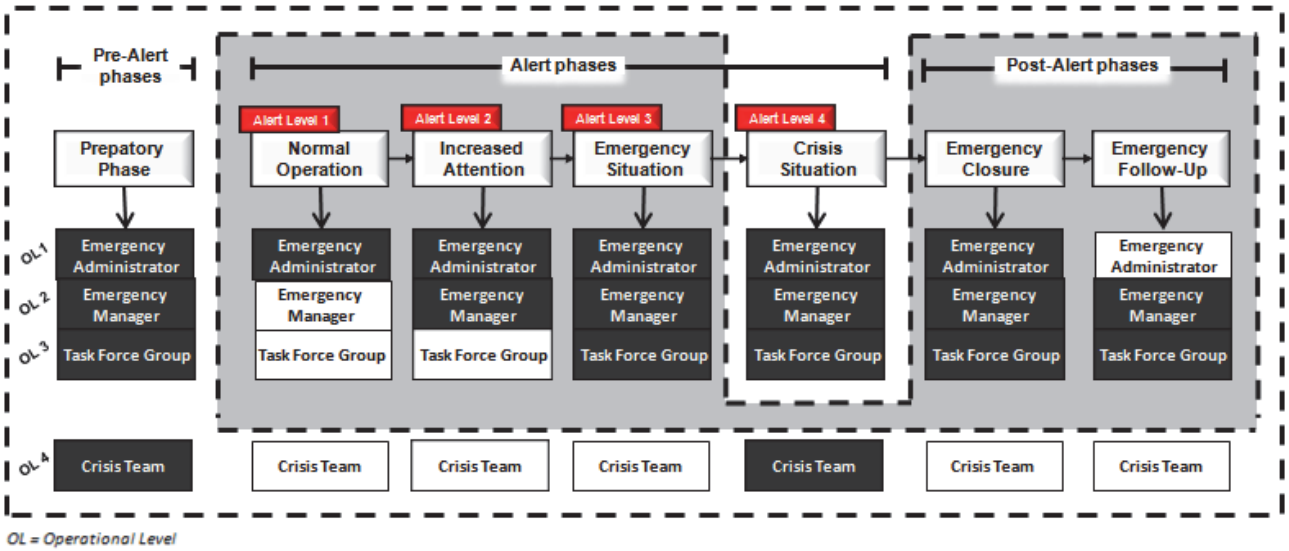


Figure 1. *CM³: Emergency Problem Management*

II. METHOD

Our journey towards evaluating *CM³: Emergency Problem Management* consisted of many stages. Unfortunately, due to space restrictions, we cannot report on them all. Our reader may however, follow them by studying our former publications that describe the initial model design [4][5][12][13][14].

In general, our work consisted of four major stages: (1) design of the initial version of *CM³: Emergency Problem Management,* (2) evaluation of the model in the context of one financial company, (3) extension of the model, and finally, (4) model evaluation within five companies.

In the first stage, we developed the initial version of *CM³: Emergency Problem Management* within SAS [4][5]. This version is demarcated by the grey shaded area in Fig. 1. It is a better structured reflection of SAS emergency process model. Its main mission is to manage emergency software problems as encountered in SAS flight booking systems. When designing it, we had many unstructured and frequent interviews and discussions with SAS emergency process owner and emergency process executors. As a next step, we compared the model to an emergency process model within *Northern Finance* [12]. We chose this company mainly due to two reasons: (1) its application domain differed from the application domain at SAS and, therefore it provided a good platform for studying the applicability of the model in a different context, (2) emergencies in the financial sector were highly time dependent where the business stake was very high and where crisis had a substantial ripple effect on other sectors of the national economy [10].

During the study of *Northern Finance* emergency process, we interviewed *Information Officer*, *Incident Handler*, *Information Security Manager*, and *Emergency Escalation Partner* in a series of consecutive interviews using four different questionnaires. All of them were very comprehensive, semi-structured and open-ended. Altogether, they consisted of 300 main questions and additional 60 follow-up questions [13].

After having studied the emergency process at *Northern Finance,* we compared it to *CM³: Emergency Problem Management,* which we then extended with several process components. In Fig. 1, they constitute the components that are not part of the grey-shaded area. They mainly concern addition of *Pre-Alert* phase, *Operational Level 4* for managing crisis (see Fig. 1). When evaluating the model, we used a semi-structured and open-ended questionnaire consisting of 106 questions. On comparison with the questionnaires used in the second stage, the questionnaire in this stage was more of a comparative character whereas the former ones were more of an explorative type.

The comparison was made within five companies. Two of these were the companies that contributed to the creation and extension of *CM³: Emergency Problem Management*. These were SAS and *Northern Finance.* Three other organizations were new organizations. These were *Loans and Bank*, *Good Things*, and *Gladstone*.

Regarding the roles interviewed, at SAS and *Northern Finance,* we interviewed the same roles anew. Regarding the remaining organizations, we interviewed different roles. At *Loans & Bank,* we interviewed their *Production Group Leader*, a role in charge of task force teams. At *Good Things,* we interviewed their *Program Manager*, the head of IT security responsible for their incident management process and their contingency management. Finally, at *Gladstone,* we interviewed a shift leader, a role responsible for coordinating and resolving the emergency situations.

III. CM³: EMERGENCY PROBLEM MANAGEMENT MODEL

CM³: Emergency Problem Management consists of six process components. They are (1) identification of the

TABLE II. EMERGENCY PROCESS ROLES

Permanent Roles		Temporaty Roles
Emergency Administrator •Focal point of contact during the entire emergency process. • Accepts and controls all the emergency problem reports • Take appropriate measures. ***Emergency Manager*** • Defines and improves emergency procedures • Assists Task Force Team in managing problems • Responsible for emergency management training ***Task Force Leader*** • Manages the resolution of the emergency problems. • Leads the Task Force Team ***Task Force Team***: • Responsible for the overall problem resolution, co-ordination of the emergency activities and tracking of the problem resolution. • Ensures that appropriate actions are taken by all the parties involved in order to re-establish the normal operation with a minimum delay. •Consists of two groups: permanent and temporary ***Permanent Task Force Group Members*** • Consists of key persons and IT management responsible for vital collaborating areas within the organisations involved.	***Crisis Manager*** • Manages and Coordinates the Crisis Management Group • Ensures that proper resources are available ***Crisis Management Group*** • Consists of business management roles and upper management • Supports the Task Force Team with business sensitive knowledge and strategic decisions • Consists of two groups: permanent and temporary ***Permanent Crisis Management Group Members*** • Consists of •Crisis Group Chairman responsible for summoning and heading the Crisis Group meeting • Crisis Communications responsible for organizing organization wide communication model. • Crisis Security responsible for security issues mainly in the context of security-critical situations. ***ADDED! Crisis Security Manager*** • Monitors, handles and coordinates staff and all types of security issues • Arranges proper physical protections ***ADDED! Crisis Communicator*** • Establishing communication routines • Supplies information to the media,	***Support Personnel*** • Consists of the support personnel on Support Line 1 and 2 • Reports on all the emergency problems to the Emergency Administrator. ***System Manager*** • Responsible for the system in which the problem was encountered. ***Developer/Maintainer*** • Responsible for changing the affected code. ***Temporary Task Force Group Members*** • Consists of System Managers, Support Personnel, Programmers, and other roles vital for resolving the emergency problem. ***Temporary Crisis Group Members*** •If needed, customer representatives and/or suppliers can partake in the Crisis Group meetings.

Added stands for roles introduced to cm[3] after this study

organisations/departments/teams involved in the emergency process, (2) products to be managed by the emergency process, (3) roles involved in conducting the emergency activities, (4) the focal point of contact through which one communicates all emergency problems, (5) the emergency process and its phases, and (6) operational levels required for handling the emergency process. Although most of these constituents are present in any process model, their presence is extremely important within the emergency process. Inefficiencies in any of them may substantially affect the process results. In this section, we present each process component and the questions that have been asked for inquiring about their credibility and usefulness within the five companies studied. The questions are presented in Table III.

A. Identification of Organization

Some software systems may be integrated with many systems that are many times evolved and maintained by several organisations. Hence, the first step when defining an emergency process should be to identify all the organisations involved in emergency situations.

To solve the problem efficiently, the collaborating companies must organize themselves and agree on and create a common emergency process model. For this reason, as indicated by Questions 18-29, we inquired about whether the companies studied involved other organizations in their respective emergency processes, and if so, whether they have agreed on a common emergency problem management process model.

B. Identification of the Product and Service Scope

Not all products and services are critical to business or safety. Therefore, as a next step, the organisations should identify the products and services to be encompassed by the emergency process. These products and services are usually safety-critical and business-critical systems.

In addition to this, the organisations should define a pertinent scale for recording the severity levels of the problems encountered and determine which severity levels should be covered by the emergency process. If the process covers several severity levels, then one should define priorities for each level and specify in what way the management of the problems with different severity levels varies. Defining severity and priority should aid organizations in taking quick and appropriate measures preventing serious ripple effects and emergency escalations.

Using Questions 15-17 in Table III, we have inquired whether the organizations have identified the scope of products and services to be covered by the emergency process. We have also asked whether they have defined severity and priority values for these products and services.

C. Designation of Roles

Designation of roles is especially important in emergency and crisis situations where conflicts of authorities, clashes over organizational domains, and organizational jurisdictional differences are common [4].

As shown in Table II, *CM*[3]*: Emergency Problem Management* identifies two groups of roles: *permanent* and *temporary*. By *permanent* roles, we mean the roles exclusively dedicated to manage the emergency situations. They are: *Emergency Administrator, Emergency Process Manager, Task Force Leader, Task Force Team,* and *Crisis Team*. By *temporary* roles, we mean the roles temporarily involved in the emergency process. They are *Support Personnel, System Users, System Managers, Temporary Task Force Group Members*, *Temporary Crisis Management Group Members* and other roles which are either responsible for the problematic system or are users of the system. Due to space restrictions, we cannot describe these roles in greater detail. We only list them and their main responsibilities in Table II. Interested readers are however most welcome to study them in [4][5][12][13][14].

 ISBN: 978-1-61208-367-4

When evaluating our model, we inquired about whether the companies studied used permanent and temporary roles within their emergency processes. We then went through *CM*[3]'s role list as presented in Table II and found out whether they were applicable within the organizations studied. At this step, we used Questions 73-82 in Table III.

D. Point of Contact

During emergencies, information flow increases drastically. If not properly managed, it may cause loss of or it may delay the delivery of important information thus substantially intensifying the problem at hand and leading to a worsened situation [9][10]. For this reason, organizations should identify ways for how the emergency problems should be reported and communicated within the organization.

An emergency problem may be encountered in various ways by various roles such as end users, system managers, external organisations, or other. Each serious problem should be immediately reported to the relevant group which constitutes a focal point of contact. One should also specify the group's availability, both within and outside the office hours. In *CM*[3], such a point of contact is provided by *Operational Level 1* conducted by the *Emergency Administrator* role (see OL 1 in Fig. 1). Please observe that *Operational Levels* are not the same as *Support Line Levels* within industry. For more information about *Support Line Levels*, we welcome the reader to study [3].

Regarding the component *Point of Contact*, we used Questions 83-87 for inquiring whether the organizations studied have defined a focal point of contact for all their emergency problems and whether they have defined its availability.

E. Process Phases

As outlined in the upper part of Fig. 1, the emergency process consists of three main phases. These are (1) *Pre-Alert Phase*, (2) *Alert Phases*, and (3) *Post-Alert Phases*. Below, we briefly describe them.

During the *Pre-Alert* phase – *Emergency Preparation*, the organizations prepare for various unforeseen emergency situations by defining or improving the emergency process, by regularly practicing it and by creating various actions and contingency plans [11]. During the *Alert* phase, the organizations attend to the encountered emergency problems. To effectively manage them, *CM*[3] distinguishes between four alert phases.

As soon as *Support Personnel* on *Support Line 1* gets a report on a serious problem, they are obliged to escalate it to the focal point of contact which is *Emergency Administrator* on *Operational Level 1* (see Fig. 1). At this moment, the problem and the process trades into the *Alert Level 1 – Normal Operation* phase. This phase only lasts for a predetermined period of time. Here, the *Emergency Administrator* collects all the information about the problem, monitors user reactions, evaluates problem severity and disseminates information to all the parties concerned.

After some predetermined period of time, the problem gets escalated to the next alert phase, *Alert Level 2 – Increased Attention*. It is now the *Emergency Manager* who becomes the owner of the problem. Together with one or several *System Managers*, he evaluates and implements possible workarounds, if any. The Emergency Manager acts as a focal point of decision.

After yet some predetermined period of time, the problem gets escalated to the *Alert Level 3 - Emergency Situation* phase. Now, the *Task Force Leader* is in charge of the emergency situation. His first action is to establish the *Task Force Team* who commonly tries to resolve the emergency problem. Finally, in cases when the problem threatens the organization's business and survival, the organization steps into the highest emergency level, which is *Alert Level 4 – Crisis Situation*. In this phase, the whole organization stands on toes including business managers and upper-level managers.

After the problem is resolved, the organization steps into the *Post-Alert* phases. Here, *CM*[3] distinguishes between two post-alert phases. These are *Post-Alert - Emergency Closure* and *Post-Alert – Emergency Follow-Up*. The *Post Emergency – Emergency Closure* phase is mainly executed by the *Task Force Leader* who writes a report on the problem and distributes it to all the parties concerned. The *Post-Alert – Emergency Follow-Up* phase, on the other hand, is conducted by the *Task Force Leader* who together with the *Emergency Manager* investigates the problem with the purpose of finding root causes underlying the problem. These causes provide an important feedback to process and product improvement.

When interviewing the companies, using Questions 30-52, we inquired whether they have defined pre-alert, alert and post-alert phases, what they do within these phases and what roles they involve.

F. Operational Levels

The whole emergency process is conducted on four operational levels (see OLs in Fig. 1). The operational levels are only defined within the context of emergency and crisis management. They do not overlap with any other organizational levels, such as for instance, *Support Line* levels [3]. However, they may strongly co-operate with them.

The designation of operational levels is very important. The process execution is strongly dependent not only on the emergency phase the process is in but also on the operational level performing it. As summarized in Fig. 1, each group of roles has clearly defined responsibilities for each phase and operational level.

1) Operational Level 1:

The *Operational Level* 1 is mainly conducted by the *Emergency Administrator*. It is involved in six process phases, the *Pre-Alert* phase, the four *Alert* phases and one *Post-Alert* phase – *Emergency Closure* (see Fig. 1). During the *Pre-Alert* phase, the individuals possessing the role of the *Emergency Administrator* exercise the emergency process and provide feedback for its improvement.

 ISBN: 978-1-61208-367-4

TABLE III. EVALUATION QUESTIONAIRE

General questions
Q1: Name: Title: Department:
Q2: What is your job description and how long have you worked within the company and with similar tasks, concerning the emergency process?
Q3: Company name: Business field:
Q4: Nr. of employees… In total: Within the IT-department, Definitions and scope of the emergency process
Q5: Does your company determine the severity of incidents?
Q6: If yes, describe which
Q7: Does your company use priority codes on problems/incidents?
Q8: If yes, describe which
Q9: Does your company differentiate between software incidents and other incidents such as hardware and/or infrastructure related?
Q10: If yes, describe which
Q11: Does your company use a structured Crisis Management process at major disasters (such as fires etc)
Q12: If so can such processes be triggered by software incidents?
Q13: Can software incidents be of business critical magnitude?
Q14: Can software incidents be of a crisis magnitude?

Product and Service scope
Q15: Does your company have specific products, services or systems that especially initiate the emergency process?
Q16: If yes, describe if there are subsets and what is included in these subsets and why?
Q17: If yes, what is not included in these subsets and why?

Organizational structure:
Q18: Are there other organizations involved in the emergency problem process?
(External maintenance or development organizations as well as suppliers and important customer that may be affected)
Q19: If yes, describe which:
Q20: Have you agreed with these organizations on a common emergency problem management process?
Q21: Are there differences in the emergency problem process depending on time of day (as in or out of office hours, different contact areas)
Q22: If yes, describe which:
Q23: Do you use task forces on different levels during emergency situations?
Q24: If so what is it called?
Q25: Are there any other groups of interest in this context?
Q26: Can you be exposed to several emergency situations simultaneously?
Q27: If so, how is this coordinated?
Q28: Are parallel solutions suggestions developed?
Q29: If so; who decides on the solution to be implemented?

Normal Operation phase
Q30: Which activities are performed during emergency initiation in the Normal Operation Phase?
Q31: Which roles are active during emergency initiation in the Normal Operation Phase and what are their responsibilities?
Q32: Who is the problem owner during emergency initiation in the Normal Operation Phase and what are his/hers responsibilities?
Q33: Are there any time frames associated with the Normal Operation phase? e.g.
1) Time limits before it are allowed to alter the system? 2) Time limits within where information must be sent out? 3) Other time limits or regulations?

Increased Attention Phase:
Q34: Which activities are performed during the Increased Attention Phase?
Q35: Which roles are active during the Increased Attention Phase and what are their responsibilities?
Q36: Who is the problem owner during the Increased Attention Phase and what are his/hers responsibilities?
Q37: Are there any time frames associated with the Increased Attention phase?

Emergency Situation Phase:
Q38: Which activities are performed during the Emergency Situation Phase?
Q39: Which roles are active during the Emergency Situation Phase and what are their responsibilities?
Q40: Who is the problem owner during the Emergency Situation Phase and what are his/hers responsibilities?
Q41: Are there any time frames associated with the Emergency Situation phase?

Emergency Closure Phase:
Q42: Which activities are performed during the Emergency Closure Phase?
Q43: Which roles are active during the Emergency Closure Phase and what are their responsibilities?
Q44: Who is the problem owner during the Emergency Closure Phase and what are his/hers responsibilities?
Q45: Are there any time frames associated with the Emergency Closure phase?

Emergency Follow-Up Phase:
Q46: Which activities are performed during the Emergency Follow-Up Phase?
Q47: Which roles are active during the Emergency Follow-Up Phase and what are their responsibilities?
Q48: Who is the problem owner during the Emergency Follow-Up Phase and what are his/hers responsibilities?
Q49: Are there any time frames associated with the Emergency Closure phase?

Preparations Phase:
Q50: Which activities are performed during emergency preparations?
Q51: Which roles are active during the emergency preparations and what are their responsibilities?
Q52: Are there any time frames associated with the emergency preparations?

Operational levels 1:
Q53: Do you have operational levels?
Q54: How many Operational levels do you have?
Q55: What is their overall function?
Q56: Operational level 1:
Q57: Which are the responsibilities of this operational level?
Q58: What activities do occur at this level?
Q59: Which roles are active in this level?
Q60: What activities are these roles involved in during Normal Operation?

Operational level 2:
Q61: Which are the responsibilities of this operational level?
Q62: What activities do occur at this level?
Q63: Which roles are active in this level?
Q64: What activities are these roles involved in during Normal Operation?

Operational level 3:
Q65: Which are the responsibilities of this operational level?
Q66: What activities do occur at this level?
Q67: Which roles are active in this level?
Q68: What activities are these roles involved in during Normal Operation?

Operational level 4:
Q69: Which are the responsibilities of this operational level?
Q70: What activities do occur at this level?
Q71: Which roles are active in this level?
Q72: What activities are these roles involved in during Normal Operation?

Roles:
Q73: How many permanent roles are involved within the emergency process?
Q74: How many temporary roles are involved within the emergency process?
Q75: Does the following roles participate in the emergency process:
Q76: • System owner/manager? Where do they reside? Temporary/permanent? Activities during normal operation?
Q77: • System specialist? Where do they reside? Temporary/permanent? Activities during normal operation?
Q78: • Business manager? Where do they reside? Temporary/permanent? Activities during normal operation?
Q79: • Business specialists? Where do they reside? Temporary /permanent? Activities during normal operation?
Q80: • Support personnel? Where do they reside? Temporary/permanent? Activities during normal operation?
Q81: • Programmers? Where do they reside? Temporary/permanent? Activities during normal operation?
Q82 • [Other roles]? Where do they reside? Temporary/permanent? Activities during normal operation?

Information flow / Point of contacts
Q83: Which focal point(s) are serious emergency problems reported to?
Q84: How is information disseminated?
Q85: Which are the target groups?
Q86: Are structured information channels set up (or already existent) during an emergency?
Q87: What are they?

Measurement Methods (and analysis)
Q88: Does your company measure the emergency process?
Q89: Exactly what do they measure?
Q90: What do you use it for?

Preparations and process improvement
Q91: Does your company practice to resolve emergency incidents (aka scenario-based training)?
Q92: Does your company perform process reviews concerning emergencies? (e.g.: Review Document Sittings.)
Q93: Does your company have technical oriented training activities concerning emergencies?
(e.g. contingency testing where the primary site is taken down and a secondary are used instead)
Q94: Does your company use any other form of training concerning emergencies?
Q95: Are there any analyses of the root causes of the emergency problem? (i.e. site-specific notes)
Q96: Is there any feedback to such analysis?
Q97: If yes, describe who
Q98: Are processes and the methods of working evaluated or analyzed?
Q99: If yes, describe which
Q100: Are there any couplings back to the process from such analysis?
(e.g. lessons learned)
Q101: If yes, describe which

Artifacts
Q102: Does your company use artifacts for providing and managing the execution of the emergency process workflows? (e.g. checklists and case management tools to support workflows according to processes)
Q103: Does your company use artifacts for operational management of a certain domain or aspect of one?
(e.g. Configuration Management databases, monitoring system, diagnosis tools).
Q104: Does your company use artifacts for contingency and recovery of business critical system?
(e.g. double systems to reduce impact and RAID and Back-Up solutions to improve system recovery)
Q105: Do these tools support the company´s working processes?
(e g adjusted for ITIL)

Extra:
Q106: Are there any other qualitative measurements used? (show the paper with qualitative crisis measures)

 ISBN: 978-1-61208-367-4

The responsibilities of the *Emergency Administrator* role vary during the *Alert* phases. In the first *Alert* phase, they own the problem. Here, they confirm the problem, establish an internal emergency log, record relevant information in it, and distribute it to all the parties concerned. The information basically specifies the problem, its occurrence, its cause, expected impact, and other relevant data.

During the remaining *Alert* phases, the *Emergency Administrator* continues administrating the problem, and informing all the parties concerned about the status of the problem. However, he no longer owns the problem. Finally, in the *Post-Alert* phases, the Emergency *Administrator* records all the problem information and informs all the parties concerned about its resolution.

The *Operational Level 2* is conducted by mainly two roles: *Emergency Manager* and *System Manager(s)*. The *Emergency Manager* has many responsibilities. One of them is to support the *Emergency Administrator* in all the emergency situations. He also coordinates workarounds received from the *System Manager(s)*. The responsibility of the *System Manager(s)*, on the other hand, is (1) to be available to the *Emergency Administrator* and the *Emergency Manager*, (2) to provide them with the necessary information and (3) to attend to the tasks requested by them.

2) Operational Level 2

The *Emergency Manager* and *System Manager(s)* start having duties on Alert *Level 2*. During the Increased *Attention* phase, the *Emergency Manager* becomes the problem owner. However, he is continuously supported by the *Emergency Administrator* with various administrative tasks. He also involves *System Manager(s)* responsible for the systems or system parts that got affected by the problem, creates workarounds in cooperation with the *System Manager(s),* and distributes information to the relevant management.

During the remaining alert phases, the *Emergency Manager* gets rid of his problem ownership. He now supports the *Task Force Team* with various tasks.

During the *Post-Alert* phases, the *Emergency Manager* continues to support the *Task Force Team*. He also evaluates the emergency process, makes suggestions for improving it and realizes them, if deemed relevant and necessary.

3) Operational Level 3

The *Operational Level 3* is mainly conducted by the *Task Force Leader* and *Task Force Team*. The roles and number of participants in the team varies depending on problem type. If, for instance, three systems are involved, then it automatically implies that three *System Managers* and their teams are involved.

The responsibilities of the *Task Force Leader* role start during the *Alert 3* phase. The *Task Force Leader* establishes a *Task Force Team* and ensures that the team is in place. Afterwards, the course of actions varies depending on the problem. However, the *Task Force Leader* acts as a focal point of entry for all the management contacts, ensures that all parties concerned are informed, leads the *Task Force Team*, co-ordinates the emergency activities, initiates activities leading to the reduction of user impact, makes sure that the initiated activities are taken according to the defined procedures, and initiates workaround's or other problem solutions.

After the problem has been resolved, the *Task Force Leader* produces a report containing (1) time when the problem first occurred, (2) description of what happened and why, (3) description of the impact, (4) measures taken to limit the impact, (5) time stamp when the problem got resolved, (6) description of the measures taken in order to resolve the problem, (7) status of the emergency procedures used, (8) action list for changes to the emergency procedures, and (9) suggestions for how to prevent similar situations.

During the *Post-Emergency – Follow-Up* phase, the *Task Force Leader* makes additional investigations of the problem and its causes together with the *Emergency Manager*. If the emergency problem is followed by a planned and scheduled problem resolution, then they should monitor its resolution.

In this phase, the *Task Force Leader* has regular meetings with the relevant roles and organisations or departments during which they follow up all problems of high severity. The goal is to find ways to avoid future emergency situations. Hence, a vital task of this phase is to specify measures to prevent the problems from occurring. These measures should be recorded and delivered to the process improvement process.

4) Operational Level 4

The *Operational Level 4* is only active in clear crisis situations. Crisis is an extraordinary situation that needs involvement from top management whose responsibility is to evaluate business threats and make important decisions on finances, personnel and other resources. It is led by *Crisis Management Group* and it is supported by *Task Force Team*. The roles and number of its participants varies depending on the crisis type.

During the interviews, using Questions 53-72, we inquired whether the organizations studied have defined their correspondences to *CM³ Operational Levels*, what roles are involved in these levels and what their responsibilities are.

IV. EVALUATION OF THE CM³: EMERGENCY PROBLEM MANAGEMENT MODEL

In this section, we present the evaluation results of *CM³: Emergency Problem Management* within the companies studied. When doing it, we follow the order of process components as defined in Section III.

A. Identification of Organisations

All the five companies have defined an emergency problem management process. However, only four of them need to involve external organizations in their emergency situations. All four of them have agreed on an emergency problem management process model to be commonly run by all the parties involved.

B. Identification of the Product and Service Scope

Only one organization identifies products and services that undergo an emergency process. It is SAS. SAS does it indirectly by classifying systems according to how soon they should be recovered. In the remaining companies, the products and services are too tightly coupled to one another

 ISBN: 978-1-61208-367-4

implying that a problem in one system might lead to a substantial ripple effect within the whole organization or even several organizations. Hence, all the products and services undergo an emergency process. The process gets enacted on the basis of an emergency case, its context, severity value and a number of the affected functions or customers.

All the organizations studied have defined severity and priority (urgency) values for their products and services. An example of how one organization formally calculates severity levels is illustrated in [12]. One of the organizations studied, however, does not have any formal definition of severity and priority. Being within online gaming industry, their severity is informally estimated by counting the number of the affected users.

Except for SAS, when enacting their emergency processes, the organizations mainly follow the urgency value and the number of the reported incidents for the emergency problem or the problem severity.

C. Emergency Management Roles

All the companies studied have defined both permanent and temporary roles. Regarding the permanent roles, all the companies have the equivalences of *Emergency Administrator, Emergency Manager* and *Task Force Leader.* However, their naming strongly differs. The role of *Emergency Administrator* is, for instance, mainly conducted by support personnel in two organizations. Other role names corresponding to *Emergency Administrator* are *Operation Manager* and *Operator* at a control department.

Regarding the role of *Emergency Manager,* we have found out that all the companies use the *Emergency Manager* role to different degrees; from providing assistance concerning problem escalation to being very active in supporting the emergency resolution process and to providing quality assurance to the *Task Force Leader*.

Regarding the role of *Task Force Leader*, all the five companies use this role as a single point of decision in the *Emergency Situation* phase. The role is primarily responsible for getting the impacted systems' functionality up and running and he has the authority to assign resources, if needed.

Four out of five organizations involve *Crisis Management Group*. The group is a meeting board responsible for the overall IT and business coordination and management. It deals with all crises related issues. It decides when to declare disaster and when to start acting according to the contingency plans.

Finally, our study has revealed the need for two additional however very important roles, *Crisis Communicator* and *Crisis Security Manager*. These roles are implemented in four of the five organizations studied. The responsibility of the *Crisis Communicator* is to manage communication on emergency problems between the organization and the public. The responsibility of the *Crisis Security Manager*, on the other hand, is to monitor, handle and coordinate staff and all types of security related issues, and to arrange proper protection

All the companies studied use temporary roles in their emergency processes, such as *Support Personnel, System Manager* and *Developer / Maintainer*. Regarding *Support Personnel*, they all have its corresponding role supporting the customers in their daily operation. It is this role that may overlap or may be merged with the role of the *Emergency Administrator*.

D. Focal Point of Contact

All the companies studied have an appointed role, or a group of roles, that act as a focal point of contact for all the emergencies. The roles involved vary. At SAS, for instance, *Operational Level 1* corresponds to the first point of contact during office hours. After a serious problem gets reported to *Support Line 1* [3], it automatically gets escalated to *Operational Level 1*. Outside office hours, however, the problem gets reported to *Support Line 1* belonging to an outsourced organization. This organization, in turn, contacts *Operational Level 1* in cases when they deem that the reported problem is serious. Regarding the remaining organizations, the *Emergency Operator* at *Operational Level 1* corresponds to *Support Line 1* being on duty around the clock [3].

E. Process Phases

Due to the fact that the organizations studied have not had any emergency standard to follow, they have defined their emergency processes on their own. For this reason, their models differ. Still, however, we could identify many common parts.

1) Pre-Alert Phase

All the organizations studied prepare themselves for various emergency and crisis situations. Hence, they have a phase corresponding to CM^3*'s Pre-Alert Emergency Preparation* phase. During this phase, they mainly review the emergency process and its supporting documents. Four out of five companies even conduct sporadic scenario-based training several times a year.

Different roles are responsible for the pre-alert emergency process within the organizations studied. They are *Contingency Manager* and various other industrial correspondences to CM^3 *Emergency Manager*.

2) Alert Phases

Reporting on serious problems/incidents to *Support Line 1* enacts the first emergency phase, *Alert Level 1 – Normal Situation,* in four of the organizations studied. At SAS, however, the problems get immediately escalated from *Support Line 1* to *Operational Level 1*, which, in turn, initiates the emergency process.

As mentioned in Section 4.3, only SAS has explicitly identified the *Emergency Administrator* role. In the other companies, the role of the *Emergency Administrator* is performed by other roles, such as support personnel or other administrative or technical roles.

In four out of five companies, the problem gets escalated to the next phase, the *Alert Level 2 – Increased Attention* phase. In these companies, the problem is now owned by the role corresponding to the *Emergency Manager* who tries to find a workaround and makes preparations for the next alert

 ISBN: 978-1-61208-367-4

phase. In the fifth company, *Good Things*, severe problems are directly escalated from *Support Line 1* to their correspondence to *CM*^{3}'s *Task Force Team* which they call *24/7 Group.*

All the companies studied have a phase corresponding to the *Emergency Situation* phase during which the *Task Force Leader* coordinates the resolution process. In three out of five companies, the first task of this role is to form a *Task Force Team.* In the other two companies, the *Task Force Leader* assigns the emergency task to one or several departments.

Regarding the *Alert Level 4 – Crisis Situation* phase, it is practiced in four out of five companies studied. Here, all business related decisions are made by the *Crisis Management Group*. This phase is triggered only in very critical business cases. The fifth company, which is a relatively young company, does not practice crisis management process yet.

Our study has revealed that the involvement of *Crisis Management Group* is immensely important in making critical decisions. Their decisions may override the decisions of *Task Force Group*, even in cases when IT solutions are more optimal than the business ones. Usually, this happens when safety or business gains are more prioritized than anything else. Scenario describing such cases is provided in [12][13].

Regarding the CM^3's suggestion for determining time period for each alert phase, only SAS does so. The other companies continuously monitor the problem during the early alert phases and escalate it to higher alert phases only if the problem and its impact intensify.

3) Post-Alert Phases

Only two companies have explicitly defined a correspondence to the *CM*^{3} *Post-Alert Emergency Closure* phase, which is conducted by the *Task Force Leader*. Just as in CM^3, the *Task Force Leader* is responsible for the follow up of the emergency cases. In the other three companies, the ownership of this phase is assigned to an *Emergency Manager* or *Root Cause Analyst.*

Irrespective of who owns the phase, all the companies studied finalize their emergency processes by having a meeting during which the problem is officially closed. In addition, three of them write and disseminate a final report on the problem and its solution.

Regarding the *Emergency Follow-up* phase, in four companies, this step is conducted by the *Task Force Leader* alone or in collaboration with other roles such as *Emergency Manager* or *Task Force Team Members*. However, the tasks defined for this phase are not always realized. Both root cause analysis and process improvement may be conducted on an ad hoc basis or they may not be conducted at all.

4) Operational Levels

Only one company has explicitly defined the operational levels as defined in CM^3. It is SAS. Regarding the remaining companies, they have done it implicitly. They follow similar levels; however, they do not call them operational levels.

All but one organization have correspondences to four operational levels. Regarding the fifth organization, as has already been mentioned, this organization is young. It has not yet managed to implement the *Crisis Situation* phase. Hence, it does not have any correspondence to *Operational Level 4*.

The scenario of defining operational levels looks as follows in the organizations studied. At their correspondences to *Operational Level 1*, support personnel, customer service representatives, or *Emergency Administrator* are the main actors. They are problem owners in the initial emergency phases, which they then hand over to the roles on the next operational level.

The main actors at the industrial correspondences to *Operational Level 2* are the *IT Support Coordinator*, *General Escalation Point* (GEP), *Emergency Escalation Partner* and *Emergency Manager* (at SAS). All these roles have the responsibilities corresponding to those of CM^3's *Emergency Manager*

All the organizations studied involve correspondences to *CM*^{3}'s *Task Force Teams* on *Operational Level 3*. Three of them actually use the same name. In one company *Task Force Leader* was called *Incident Handler*. Only two companies use different names such as *24/7 Group* and *Shift Leader*.

Regarding the industrial correspondences to *Operational Level 4*, as mentioned earlier, only four organizations have implemented it in their process models. They use the same role names as in CM^3. One company, however, calls the CM^3's correspondence to *Crisis Manager* as *Critical Situation Manager (CSM).*

V. CONCLUSION

Due to the fact that the software community lacks a common emergency maintenance process model, many organizations have defined their own local emergency process models. In this paper, we have studied five industrial emergency maintenance processes with the purpose of evaluating the applicability of *CM*^{3}*: Emergency Problem Management* within five companies. The companies running these processes differ in size, industrial domains and process maturity. Despite this, with the use of an open-ended questionnaire and *CM*^{3}*: Emergency Problem Management*, we could identify their common parts and directly map them on *CM*^{3}*: Emergency Problem Management.* Here, the *CM*^{3} model has acted both as a helpful tool for evaluating industrial emergency process models as well as an excellent tool for evaluating itself and its structure. It has helped us to find many commonalities on how to meet emergency situations and it has helped us to identify some minor differences among the processes studied. Below, we briefly list our findings, comment on them and comment on how they contributed to enhance the quality of *CM*^{3}*: Emergency Problem Management.*

All the organizations studied have defined an emergency problem management process to be either used locally for managing their internal emergencies or as a common process to be used together with their partners. Hence, they constitute an appropriate forum for evaluating *CM*^{3}*: Emergency Problem Management*.

- Not all the organizations identify the scope of their product and service portfolios that might be subdue

 ISBN: 978-1-61208-367-4

to emergency problem management. A strong coupling among the systems and magnitude of the potential ripple effect makes four out of five organizations be very sensitive to all types of emergency problems in all their systems. Hence, we conclude that the design of products and services in these companies is not amenable for defining and enacting the emergency process. The organizations must be on a constant alert about all types of incidents that are encountered in all their products and services. This is not an effective way of managing organizational resources. As a remedy, we suggest that the organizations studied make effort in decoupling their critical systems so that the emergency process may be isolated to a specific system or even system part.

- Despite process differences in the organizations studied, all the organizations have defined software emergency process models that consist of pre-alert, alert and post-alert phases and that include activities and responsibilities that are organized in a similar manner as CM^3's operational levels. [4][5][12][14] However, the number and names of their alert phases may vary. The pre-alert and alert activities are actively conducted whereas the post-alert activities such as collecting lessons learned were sparse, and were usually only conducted in an ad hoc manner. While studying the stages, we realized that the pre-alert stage of CM^3: Emergency Problem Management needs to be explored more in depth.
- Two of the companies have defined an additional emergency operational level, the level dealing with crisis management. The other companies had a crisis management processes, but this process was not aligned with the emergency process. Crisis management is used only in cases when a software problem jeopardizes human life and/or company's financial position or survival. For this reason, we have enhanced *CM^3: Emergency Problem Management* with a crisis phase, *Crisis Situation*, on top of the emergency phase and added an additional operational level, Operational Level 4, the level only dealing with crisis management.
- Involving crisis management is more common in financial and aviation sectors than in other sectors. Still, however, the organizations studied have not been able to optimally integrate crisis management process with software problem management process. By not having an integrated crisis management process, a set of issues is raised when the two processes work side by side: (1) how to deal with single point of decision and (2) how to deal with a focal point of information during high priority emergency situations. At the moment of writing this paper, SAS is in the process of connecting the emergency incident process with the business crisis management process.
- All of the companies have implicitly defined actions to meet a software emergency situation, and these actions were conducted by a number of predefined emergency roles. These roles are either temporary or permanent emergency maintenance roles. However, four out of five have a clearly defined crisis management group. In our study, we have identified new roles such as *Crisis Management Group, Crisis Manager, Crisis Communicator*, and *Crisis Security Manager*. All these roles have been added to the CM^3 model due to its extension with an additional alert phase, *Crisis Situation*, and an additional operational level, *Operational Level 4*.
- All the companies had also identified focal points for the information flow to and from the emergency team. In all cases, it is support personnel that accepts emergency problem reports and either continues managing them or hands them over to *CM^3*'s correspondence to *Emergency Administrator.*
- Regarding *CM^3*'s suggestion for determining time period for each alert phase, only SAS does so. They do so because they have specified rules for how soon their systems should be up and running. The other companies continuously monitor the problem during the early alert phases and escalate it to higher alert phases only if the problems and their impact intensify. This is because not all problems are directly recognized as very serious and urgent. To make our model adaptable to this new finding, we change the escalation rules from only time-dependent to both time and impact dependent.
- Most of the companies conduct post-alert phases mainly on an ad hoc basis. Reasons are many. One of them is the fact that the organizations do not designate enough resources for this important phase. Another reason is the fact that the report on the emergency problem and measures is disseminated too late. Its receivers lose interest in taking any measures whatsoever due to new problems that they have to deal with instead.
- Only one company has explicitly defined operational levels. The other companies have implemented operational levels implicitly by defining operational responsibilities and tasks and making sure that they do not overlap across the roles involved in emergencies.

Our evaluation study was huge. Hence, we could not present all our findings. We only had to concentrate on the most important ones. Using them as a basis, we may claim that *CM^3: Emergency Problem Management* is applicable within the industry. There are many commonalities between *CM^3: Emergency Problem Management* and the industrial emergency process models studied. We believe that our work on CM^3: Emergency Problem Management shows evidence for the presence of software emergency processes and the need for a standard that can aid practitioners in setting up and evaluating their local processes.

Our work on CM^3 is still in an early stage. Due to the fact that the emergency process is very comprehensive and complex, more studies are needed to fully evaluate the

 ISBN: 978-1-61208-367-4

model. In brief, the following research action points need to be considered:

- The pre-alert phase needs to be further investigated. Two actions are proposed: 1) to survey training and education efforts, and 2) to explore how lessons learned from previous incidents can be used as feedback into the emergency maintenance process.
- Evaluate *CM³: Emergency Problem Management* within other industrial sectors such as, for instance, health care and e-government. Due to their nature, potential emergencies can be disastrous in these fields.
- Coupling *CM³: Emergency Problem Managemen*t with crisis management. Several issues are of interest such as mapping a single point of technical decisions from the emergency process onto a single point of organizational decisions from the crisis management and vice versa and define enterprise-wide agreements on when to declare a crisis situation.
- Integrate *CM³: Emergency Problem Management* with the development phases of the software lifecycle, identify how they impact each other and clarify borders between software emergency maintenance and other processes such as risk management, scheduled problem management and the like.

Despite many action points required for evaluating the model, we strongly believe that *CM³: Emergency Problem Management* already provides solid guidance for software organizations in their attempts to define and improve their emergency software maintenance process models.

REFERENCES

[1] A. Brown and D. Patterson, "Embracing failure: A case for Recovery-Oriented Computing (ROC)". In Proceedings of the 2001 High Performance Transaction Processing Symposium, 2001, pp. 3-8.

[2] M. Kajko-Mattsson, “Motivating the Corrective Maintenance Maturity Model (CM3)”, In Proceedings of the Seventh IEEE International Conference on Engineering of Complex Computer Systems, IEEE, 2001, pp. 112-117.

[3] M. Kajko-Mattsson, Corrective Maintenance Maturity Model: Problem Management, PhD thesis, ISBN Nr 91-7265-311-6, ISSN 1101-8526, ISRN SU-KTH/DSV/R--01/15, Department of Computer and Systems Sciences (DSV), Stockholm University and Royal Institute of Technology (KTH), 2001.

[4] M. Kajko-Mattsson, C. Nielsen, P. Winther, B. Vang and A. Petersen, "An Outline of CM3: Emergency Problem Management," In Proceedings of EUROMICRO Conference on Software Engineering and Advanced Applications, 2005, pp. 292-303.

[5] M. Kajko-Mattsson, C. Nielsen, P. Winther, B. Vang and A. Petersen, 2006. "Eliciting CM3: Emergency Problem Management at Scandinavian Airline Systems", Journal of Research and Practice in Information Technology, 2006, vol. 38, no. 4, pp. 303-316.

[6] D. L. Parnas, A. J. van Schouwen and S. P. Kwan, "Evaluation of Safety-Critical Software" Communications of the ACM, vol. 33, no. 6, pp. 636-648, June 1990, doi:10.1145/78973.78974

[7] T.C. Pauchant, and R. Douville, 1993. "Recent research in Crisis Management: a study of 24 authors’ publications from 1986 to 1991", Organization & Environment, vol. 7, no. 1, pp. 43-66, Jan 1993, doi: 10.1177/108602669300700104.

[8] C. M. Pearson and J. A. Clair, "Reframing Crisis Management", Academy of Management Review, vol. 23, no. 4, pp. 59-76. Academy of Management, 1998, doi: 10.5465/AMR.1998.192960.

[9] E.L. Quarantelli, "Disaster Crisis Management: A summary of research findings", Journal of Management studies, vol. 25, no. 4, pp. 373-384, 1988. ISSN: 0022-2380

[10] A. Reilly, "Preparing for the worst: the process of effective crisis management" in Organization Environment, vol. 7, no. 2, pp. 115-143, Jan 1993, doi:10.1177/108602669300700204.

[11] D. Smith, "Beyond contingency planning: towards a model of crisis management", Industrial Crisis Quarterly, vol. 4, pp. 263-275, Jan 1990, doi:10.1177/108602669000400402.

[12] J. Snygg, M. Kajko-Mattsson, and E. Hammargren, Comparing two software emergency process models. In: 2012 International Conference on Software and System Process (ICSSP). Zürich: IEEE Conference Publications, 2012 pp.150-159

[13] J. Snygg and E. Hammargren, 2010. “Handling Crisis Within CM3: Emergency Management Process” master thesis, Dept. of Computer and Systems Sciences., Stockholm University/Royal institute of Technology., 2010.

[14] M. Kajko-Mattsson, J. Snygg, and E. Hammargren, CM3: Emergency problem management - A scenario-based evaluation. In: Information Science and Digital Content Technology (ICIDT), 2012 8th International Conference on Jeju Island, Korea (South): IEEE, 2012, pp.379-386

[15] ISO/IEC 14764., (IEEE, std 14764-2006). Software Engineering-Software Life Cycle Process-Maintenance, The Institute of Electrical and Electronics Engineers, Inc., 2006.

[16] M. Kajko-Mattsson, Common Concept Apparatus within Corrective Software Maintenance, Proceedings, International Conference on Software Maintenance, IEEE Computer Society Press: Los Alamitos, CA, Sep 1999, ISBN: 0-7695-0016-1, pp. 287-297.

 ISBN: 978-1-61208-367-4

An Analysis of Domain and Application Engineering Co-evolution for Software Product Lines based on Cladistics: A Case Study

Anissa Benlarabi
IMS Team, SIME Laboratory
ENSIAS, Mohamed V Souissi University
Rabat, Morocco
a.benlarabi@gmail.com

Amal Khtira
IMS Team, SIME Laboratory
ENSIAS, Mohamed V Souissi University
Rabat, Morocco
amalkhtira@gmail.com

Bouchra El Asri
IMS Team, SIME Laboratory
ENSIAS, Mohamed V Souissi University
Rabat, Morocco
elasri@ensias.ma

***Abstract*—Software product line engineering is a discipline for large scale reuse, its main advantage is the ability to reuse a set of domain assets in the development of a large number of products. In order to achieve this benefit, the software product line must cope with business requirements evolution. When dealing with evolution, the most effort must be granted to the understanding of the change and the identification of its impact because changes happening to products must be propagated to domain artifacts that are used for the whole family, and if the impact is not studied, each product will evolve separately from the domain assets. Many techniques were proposed to facilitate the impact analysis, such as evolution traceability or documentation. However, they consider only the change on the domain assets level and they underestimate issues raised by the fact when products evolve separately from the domain assets, which decreases the ability of the software product line to derive all the products features. In this paper, we tackle this issue by analyzing the co-evolution of software product lines and their products. We use cladistics classification, which was used in biology to construct their evolutionary trees, then we compare the trees using mathematical analysis and we propose a solution to restore the perfect co-evolution of the software product line and its products. We carried out a case study on a Mobile Media software product line to illustrate our approach.**

***Index terms*—** Software product lines; Co-evolution; Cladistics.

I. Introduction

Software product line engineering [1] is a software engineering discipline centered on reuse. It consists in developing a set of domain assets, which can be reused to derive a set of products for a particular market [2]. Its main goal is the reduction of costs and time to market, which can only be achieved by the continuous adaption of the domain assets to the ever-changing user requirements. Hence, to maximize benefits from the software product line common platform, the evolution activity must be the pivot activity of the software product line process development.

The primary aim for the evolution activity is the protection of the software from the aging problem, which pictures the fact of having a vital software for the organization but which cannot be evolved [3]. Unlike single software, software product line aging problem is not only caused by the loss of knowledge but also by the inability of the software product line to support all the features of the old and the new products. This happens especially when the changes happening to the products are not propagated to the domain artifacts, in this case each product evolves separately from the domain artifacts and the software product line will no longer be able to derive all the features. Hence, instead of having a software product line we will have a set of independent products.

The approach presented here aims at improving the understanding of how the software product line and its products evolved in time and how they influenced each other during their evolution. It focuses on analyzing the co-evolution of the software product lines and their products. The change in software product lines has two levels, the level of domain engineering and the level of application engineering, the evolution of each level impact the evolution of the other, our co-evolution analysis helps identifying how the evolution of products and the evolution of the core assets impacted each other. In this paper, we focus on the impact of the changes happening to products on the core assets because it was less tackled by the researchers than the domain engineering evolution impact. Co-evolution was extensively studied in biology [4] to show how organisms influence each other during their evolution. The co-evolution of host-parasite is a famous example from biology [5]. Beside biology, the co-evolution was studied also in software engineering [6] [7] [8].Similarly to co-evolution in biology, we will study the co-evolution of many populations of software. Our work consists in a co-evolution model for software product lines based on cladistics classification [9], which identifies the evolution path of a group of organisms based on their shared characters and classifies them in evolutionary tree. We start by establishing the evolutionary trees of the software product line and its products, then we perform a mathematical analysis to correct divergences between their evolution paths. We illustrate our approach through a case study on the mobile media software product line [10], we started by applying the approach on one product but we intend to experiment it on other products and compare the obtained results. Currently, we consider that all the products features must be derived from the domain engineering; we do not consider the products specific features that are not intended to be part of the platform.

In Section 2, we explain the co-evolution in biology, the we present some co-evolution studies in software engineering and we introduces the co-evolution of domain and application assets. In Section 3, we propose present our approach through a case study; we firstly study the evolution courses of the

 ISBN: 978-1-61208-367-4

software product line and its products through their evolutionary trees established using cladistics classification. Secondly, we compare these resulted trees to extract their similarities and divergences then we correct these divergences using a mathematical analysis. We give a conclusion in Section 4.

II. Related Works

According to our literature search, works done to understand how the evolution of domain engineering and the evolution of application engineering influence each other rely mainly on traceability links between the artifacts of the two levels. A framework for traceability was proposed by Anquetil et al. [11], the framework allows for tracing links between the different artifacts and present them in a graphical view, the developers can use the graphical view to know the impacted artifacts by a change. Ajila and Kaba [12] proposed a tool which gives operation instances modification, operations for consistency checking, and operations for change impact analysis. The tool calculates the impact of a change on the basis repositories that involved the software product line artifacts and their relationships. Goknil proposed a meta-modeling approach for requirements traceability management [13]. He focuses on post-requirements traceability, in particular between requirements models and architectural models, the goal is to determine which architectural components are impacted by a requirement change. Traceability approaches consider the traceability of links between the domain assets and the links between the domain and the application assets as a basis for the change impact analysis activity. However, they rely on human knowledge which is too expensive and error prone. In addition, they consider that the change happens only in the domain assets. Our work allows for defining the impact of the change by identifying the hidden links between the reference and the application assets, it also improves the change understanding through a synthesis of the history of the software product line evolution and help predicting future changes by considering changes that were implemented at a product level and may be propagated to the reference assets and then to the other products. Instead of relying on the human knowledge, we use the evolution histories of the software product line and its products and we analyze their co-evolution using cladistics classification.

III. Software Product Lines Co-evolution

In this section, we present the co-evolution principal, which was used mainly in biology, and we give an insight on some works that deal with the co-evolution in software engineering context. Thus, we introduce the co-evolution of domain engineering and application engineering in software product lines.

A. Co-evolution in Biology

Co-evolution of species in biology describes the situation when an evolution of a population of species can affect another population of species, and consequently induces its evolution. It consists in a mutual evolutionary influence between two populations [4]. A population in biology represents any group of descendants of the same ancestor that appeared due to changes of the ancestor characteristics. Understanding how populations co-evolve allows for determining how environmental changes impact directly their evolution. The co-evolution of host-parasite is a famous example of biological co-evolution. Because parasites cause damages to their hosts, hosts develop new capacities to resist to their parasites however parasites also develop capacities to overcome this resistance [5]. Therefore, a clearer understanding of hostparasite co-evolution will point to new possibilities for organic farming and reduce the application of ecologically harmful chemicals.

B. Co-evolution in Software Engineering

Co-evolution was tackled in other fields, such as software engineering; we present here some works that showed the necessity to take into consideration the co-evolution between different layers of a solution to preserve its consistency and correctness and also to reduce evolution costs.

Ruscio et al. [6] addressed the co-evolution of meta-models and their related entities: models, transformations and tools, especially the automated adaptation of these entities in order to preserve their correctness and consistency. The authors introduced a set of basic ingredient a co-adaptation solution must provide, and they point out on the necessity to have a unique technique for meta-models co-evolution regardless the related entity type. They proposed the EMFMigrate tool, which applies a set of migration rules on the related entities depending on the change type and the relation between the metamodel and the entity, because in some relations meta-models changes may be independent and do not require a co-evolution.

Kster and Trifu [7] tackled the problem of traceability between the requirements and the architecture incited by the fact that an important part of evolution costs are spent to locate the impacted elements. He presents a case study on the co-evolution between requirements and architectural design from which he extracted a set of requirements for a solution of co-evolution of architectural model and requirements model. Then he proposed a solution using graphs in which elements from both models are linked by decisions. The graph is dynamically navigable, and helps identifying the change impact easily.

Seidl et al. [8] introduced the co-evolution of software product lines. He stated that evolution of SPLs can harm the mapping between features and realization artefacts, for example if an implementation asset is deleted and a mapping to it remains in the system, products that include features mapped to this missing item will be invalid. For this reason, proposed an approach to co-evolve the features mapping and the system models, more accurately the feature model and the realization artefacts. He made a classification of evolution scenarios either in problem space (insert feature, delete feature, Split feature, etc.) or in solution space (replace method, rename method, etc.) into two groups: interspatial evolutions that reaches beyond the boundaries of the originating space and intraspatial

 ISBN: 978-1-61208-367-4

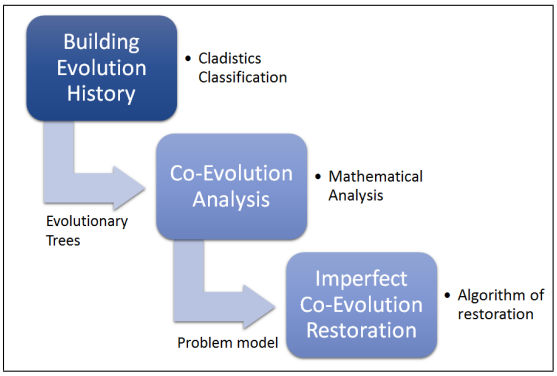


Figure 1. Co-evolution approach for SPL

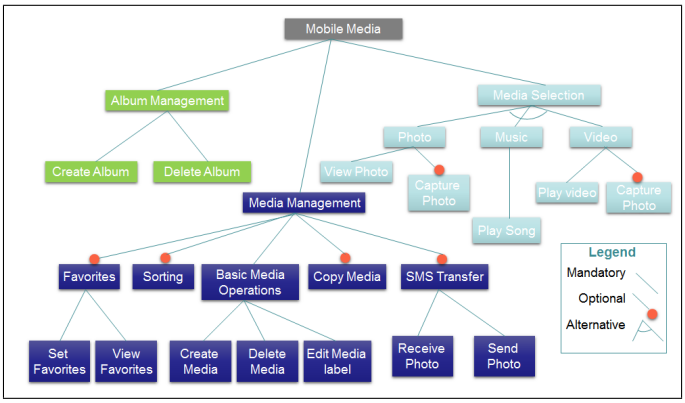


Figure 2. Mobile Media SPL features model

evolutions that impact only the space they are originating from. Thus, he extended Eclipse by a set of remapping operators to maintain the consistency of features mapping. These operators will be sequentially executed after each interspatial evolution.

C. Domain and Application Assets Co-evolution in Software Product Lines : a cladistics based approach

In this paper, we introduce the concept of co-evolution of domain and application assets in software product lines, which consists in comparing the evolution paths of the domain assets and the application assets and then deducing if application assets were changed independently from their domain assets. Organisms co-evolution analysis relies mainly on the visible characters of these organisms, in software the visible characters are its features. Hence, we will consider only features models; thus, we study the co-evolution of the domain features model and the application features models.

To deal with such co-evolution, we propose an approach based on Cladistics [9], which is a biological technique used to understand how organisms evolve over time (see Fig. 1.), it builds an evolutionary tree for a population by classifying its members in a tree on the basis of the evolution of their physical characters or the evolution of their behavior. The steps of our approach are as follows:

- Building evolution history: in this step, we use the data about evolution in order to establish the evolution path of the software product line and each derived product. We use cladistics classification, which gives a classification of the members of a population based on their shared characters. In the case of software product lines, the characters of software are its features
- Co-evolution Analysis: in this step, we perform a mathematical analysis of the domain and applications assets co-evolution through sets, we introduce the hypothesis of our analysis and we represent the perfect co-evolution by means of mathematical equalities
- Imperfect co-evolution correction: in this step, we present mathematically the imperfect co-evolution on the basis of the analysis we did in step 2 and we propose an algorithm to correct.

In the following subsections, we will explain in more details our approach and the techniques used in each step.

IV. Domain and Application Engineering Co-evolution Analysis: a Case Study

In this section, we present our approach co-evolution approach through a case study on the mobile media software product line with one of its derived products. In the first subsection, we present the mobile media software product line and the derived product features, in the second subsection, we construct their cladograms using cladistics classification, a cladogram is a branching diagram which represents the evolution path of a group of organisms based on their shared characters. In the third section, we compare their cladograms and we correct the detected imperfections using a mathematical analysis.

A. Mobile Media software product line

The Mobile Media software product line manipulates photo, music, and video on mobile devices, such as mobile phones and it has 200 derived products [10]. Many evolution scenarios were performed on the mobile media software product line, we take into consideration in our case study six evolution scenarios. Hence, we have a population P1 formed by seven releases of the software product line and a population P2 formed by the seven releases of a derived product. We present the feature model of the seventh release of the mobile media software product line in Fig. 2.

B. Building Evolution History

In order to build the evolution history for the populations of the software product line and the product, we use cladistics classification, which is a biological technique which classifies a set of organisms derived from the same ancestor in a evolutionary tree. In the following subsections, we give more details about this technique and its application on the mobile media software product line.

1) Cladistics Classification: Cladistics classification was used in biology to construct evolutionary trees that shows the evolutionary relationships among various biological organisms, on the basis of the similarities and differences in

 ISBN: 978-1-61208-367-4

their physical or genetic characteristics [9]. It assumes that in a population of organisms, a new organism appeared due to the change of the group characters. Hence, it identifies the evolution path of these organisms based on their shared characters. In the case when more than one possible tree can be generated for one group we must choose the most parsimonious evolutionary tree which is the shortest one. The length of a tree is obtained by calculating the sum of all the characters fits where the fit of a character is the number of its occurrences on the tree. In addition to identifying the evolution path of a taxonomic group, cladistics classification helps identifying which character change is responsible for the appearance of each organism and also the characters that mostly participate to the evolution of the group. The steps of the cladistics classification are:

- Select the population to be classified
- Identify the characters of the population and their different states
- Classify the group on the basis of their shared characters in an evolutionary tree called cladogram
- When having more than one cladogram in result, an analysis of parsimony is required.

2) Evolutionary trees of mobile media and its product: We follow the mentioned steps for the cladistics classification.

The first step consists in defining populations for which we will study the co-evolution. a population is constructed by a set of organisms derived from the same ancestor by adding new characters or capabilities. in order to compare the evolution path of the software product line mobile media with the evolution path of its derived product, we must build their evolution paths. We constructed two populations, the first one P1 is formed by seven versions of the software product line mobile media, while the second one P2 is formed by seven versions of the derived product. In Tables 1 and 2, respectively, we give a detailed description of the populations P1 and P2, which we constructed on the basis of the feature models of the different releases of P1 and P2:

After we defined the two populations P1 and P2, the second step consists in defining the characters of each population. In biology, the characters of organisms of a population represent their visible traits, which could be physical or behavioral characteristics. Hence, for each population, we will identify its behavioral characters that are its features. By assuming the hypothesis $H0$ of features Independence, the set of characters of a population will be composed by independent features. In Tables 1 and 2, respectively, we formulated the vectors of features of the populations P1 and P2 as follow, the number of features are 19 and 16 for $A1$ and $A2$, respectively:

$$A1 = \{F_{1,i} \text{ while } i \in N, i \leq 19\},$$
$$A2 = \{F_{2,i} \text{ while } i \in N, i \leq 16\}$$

In order to classify the versions of each population, we will construct in the third step the features states matrices that illustrate the states of features in each version. Each feature has two states, the primitive state, which denotes the nonexistence of the feature and it is represented by 0, and the derived state, which denotes its existence and it is represented by 1. The features state matrices of our populations P1 and P2 are presented in Tables 3 and 4, respectively. We construct cladograms on the basis of these matrices by grouping versions together based on their shared characters. In this steps we used the tools PHYLIP to generate the coordinates of the evolutionary trees of P1 and P2 from their features state

TABLE I. MOBILE MEDIA SOFTWARE PRODUCT LINE POPULATION

Version	Description
V1.0	The first release of the mobile media software product line, this release encompasses the following features: Manage photos ($F_{1,1}$), Create album ($F_{1,2}$), Delete album($F_{1,3}$), Create media ($F_{1,4}$), Delete media ($F_{1,5}$), View media ($F_{1,6}$), Sort media ($F_{1,7}$), Edit media label ($F_{1,8}$)
V1.1	The second release of the mobile media software product line, in which the following features were added: Set favorites ($F_{1,9}$) and See favorites ($F_{1,10}$)
V1.2	The third release of the mobile media software product line, in which the feature Copy media ($F_{1,11}$) was added
V1.3	The fourth release of the mobile media software product line, in which the following features were added: Send media ($F_{1,12}$) and Receive media ($F_{1,13}$)
V1.4	The fifth release of the mobile media software product line, in which the feature Add music media management ($F_{1,14}$) was added
V1.5	The sixth release of the mobile media software product line, in which the following features were added: Add video media management ($F_{1,15}$), Capture videos ($F_{1,16}$) and Capture photos ($F_{1,17}$)
V1.6	The seventh release of the mobile media software product line, in which the following features were added: Play videos ($F_{1,18}$) and Play music ($F_{1,19}$)

TABLE II. DERIVED PRODUCT POPULATION

Version	Description
V2.0	The first release of the product, this release encompasses the following features: Manage photos ($F_{2,1}$), Create album ($F_{2,2}$), Delete album($F_{2,3}$), Create Photo ($F_{2,4}$), Delete Photo ($F_{2,5}$), View Photo ($F_{2,6}$), Sort media ($F_{2,7}$), Edit media label ($F_{2,8}$)
V2.1	The second release of the product, in which the following features were added: Set favorites ($F_{2,9}$) and See favorites ($F_{2,10}$)
V2.2	The third release of the product, in which the feature Copy media ($F_{2,11}$) was added
V2.3	The fourth release of the mobile media software product line, in which the following features were added: Send media ($F_{2,12}$) and Receive media ($F_{2,13}$)
V2.4	The fifth release of the product, in which the feature Print photo ($F_{2,14}$) was added
V2.5	The sixth release of the product, in which, the feature Capture photos ($F_{2,15}$) was added
V2.6	The seventh release of the product, in which, the feature Share photo in social websites ($F_{2,16}$) was added

 ISBN: 978-1-61208-367-4

TABLE III. FEATURES STATES MATRIX B OF THE MOBILE MEDIA SPL

B	$F_{1,1}$ .. $F_{1,8}$	$F_{1,9}$ $F_{1,10}$	$F_{1,11}$	$F_{1,12}$ $F_{1,13}$	$F_{1,14}$	$F_{1,15}$ .. $F_{1,17}$	$F_{1,18}$ $F_{1,19}$
V0	1	0	0	0	0	0	0
V1	1	1	0	0	0	0	0
V2	1	1	1	0	0	0	0
V3	1	1	1	1	0	0	0
V4	1	1	1	1	1	0	0
V5	1	1	1	1	1	1	0
V6	1	1	1	1	1	1	1

TABLE IV. FEATURES STATES MATRIX C OF THE PRODUCT

C	$F_{2,1}$ .. $F_{2,8}$	$F_{2,9}$ $F_{2,10}$	$F_{2,11}$	$F_{2,12}$ $F_{2,13}$	$F_{2,14}$	$F_{2,15}$	$F_{2,16}$
V0	1	0	0	0	0	0	0
V1	1	1	0	0	0	0	0
V2	1	1	1	0	0	0	0
V3	1	1	1	1	0	0	0
V4	1	1	1	1	1	0	0
V5	1	1	1	1	1	1	0
V6	1	1	1	1	1	1	1

matrices. In Fig. 3. we present the example of the input file of P1. The output file of Phylip contains the coordinates of the evolutionary tree of the population P1, we used the online tool Phyfi which we present in Fig. 4 in order to compile this file and generate the cladogram. The resulted cladograms of P1 and P2 are illustrated in Fig. 5 and Fig. 6, respectively.

C. Co-evolution Analysis

In biology, the perfect co-evolution can be restored by identifying the branches that cause this divergence and extending the cladograms by them. However, by assuming the hypothesis $H1$ that features of the software product line are sufficient but not necessary to derive all the features of the derived products, we will eliminate the imperfection caused by branches that exist in the software product line cladogram and are absent from the products cladogams. Our hypothesis is based on the fact that the software product line feature model take into consideration commonality and also variability of products. In this subsection, we will formulate our hypothesis about the perfect co-evolution in software product lines. Thus, we verify

```
1      7    19
2  V1.0     1111111100000000000
3  V1.1     1111111111000000000
4  V1.2     1111111111100000000
5  V1.3     1111111111111000000
6  V1.4     1111111111111100000
7  V1.5     1111111111111111100
8  V1.6     1111111111111111111
```

Figure 3. Input file for drawing the cladogram of P1

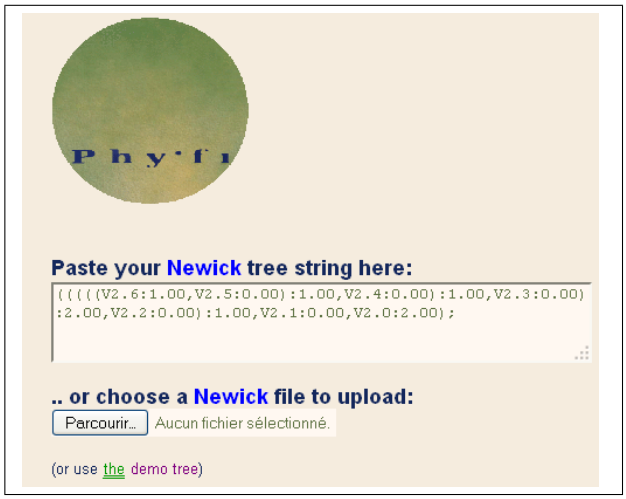


Figure 4. The coordinates of the cladogram P1

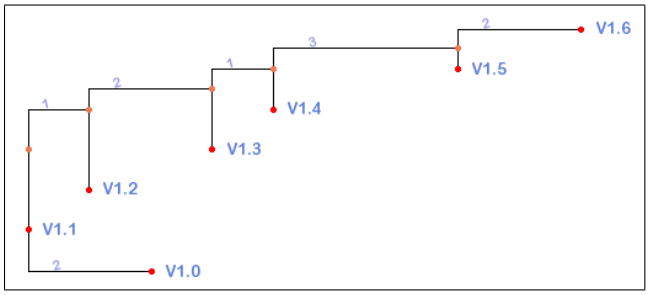


Figure 5. The cladogram of the Mobile Media SPL

these hypothesis for the two populations and we propose an algorithm to correct the extracted imperfections

1) Perfect co-evolution modeling: We set three hypothesis for the software product line, $H0$ and $H1$ are already explained above, in addition we formulated a new hypothesis $H2$ on the basis of $H0$ and $H1$:

(H0) features independence: In the set of features $A1$ and $A2$, the features are independent from each other

$$\forall i, j \in N, i \leq 19, j \leq \text{n}, F_{1,i} \neq F_{1,j}$$
$$\forall i, j \in N, i \leq 16, j \leq \text{m}, F_{2,i} \neq F_{2,j}$$

(H1) domain features sufficiency: Each feature of the set $A2$ has a corresponding feature in the set $A1$

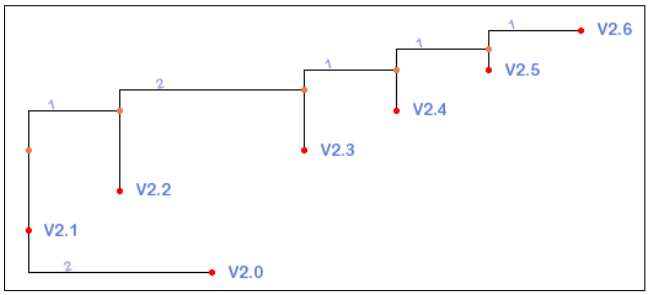


Figure 6. The cladogram of the derived product

$$\forall j \in N, j \leq 16, \exists i \in N, i \leq 19 / F_{2,j} = F_{1,i}$$

(H2) features exclusion: This hypothesis is deduced from the combination of **H0** and **H1**. Each feature of $A2$ has only one corresponding feature in $A1$

$$\begin{cases} F_{2,i}, F_{2,j}/i, j \leq 16\} \subset \{F_{1,x}, F_{1,y}/x, y \leq 19\} \\ F_{2,i} = F_{1,x} \end{cases} \Rightarrow F_{2,j} = F_{1,y}$$

On the basis of the hypotheses **H0** and **H2**, we deduce that the relationship between the two cladogram of P1 and P2 must respect the following inequality:

$$B \times A1 \geq C \times A2 \quad (1)$$

This inequality means that for each couple of leafs $L1i, L2i/0 < i \leq \mathrm{k}$ of the cladograms of P1 and P2, the number of features of $L1i$ must be superior to the number of features of $L2i$. Assuming the hypothesis **H1**, the inequality can be reduced to the following equality. The vector $A3 = \{F_{3,i}$ while $i \in N, i \leq 3\}$ represents the features of the software product line that are not supported by the derived product, and the entries di, j of the matrix D are equal to 0 or 1 depending on weather the features exist in the product or no :

$$B \times A1 - C \times A2 = \begin{pmatrix} d_{1,1} & \cdots & d_{1,s} \\ d_{2,1} & \cdots & d_{2,s} \\ \vdots & \ddots & \vdots \\ d_{k,1} & \cdots & d_{k,s} \end{pmatrix} \times \begin{pmatrix} F_{3,1} \\ F_{3,2} \\ \vdots \\ F_{3,i} \\ \vdots \\ F_{3,s} \end{pmatrix} \quad (2)$$

After the calculation of this equality we will obtain seven equalities as follow:

$$\begin{cases} \sum_{i=1}^{19} b_{1,i} \times F_{1,i} - \sum_{j=1}^{16} c_{1,j} \times F_{2,j} = \sum_{j=1}^{3} d_{1,j} \times F_{3,j} \\ \sum_{i=1}^{19} b_{2,i} \times F_{1,i} - \sum_{j=1}^{16} c_{2,j} \times F_{2,j} = \sum_{j=1}^{3} d_{2,j} \times F_{3,j} \\ \sum_{i=1}^{19} b_{3,i} \times F_{1,i} - \sum_{j=1}^{16} c_{3,j} \times F_{2,j} = \sum_{j=1}^{3} d_{3,j} \times F_{3,j} \\ \sum_{i=1}^{19} b_{4,i} \times F_{1,i} - \sum_{j=1}^{16} c_{4,j} \times F_{2,j} = \sum_{j=1}^{3} d_{4,j} \times F_{3,j} \\ \sum_{i=1}^{19} b_{5,i} \times F_{1,i} - \sum_{j=1}^{16} c_{5,j} \times F_{2,j} = \sum_{j=1}^{3} d_{5,j} \times F_{3,j} \\ \sum_{i=1}^{19} b_{6,i} \times F_{1,i} - \sum_{j=1}^{16} c_{6,j} \times F_{2,j} = \sum_{j=1}^{3} d_{6,j} \times F_{3,j} \\ \sum_{i=1}^{19} b_{7,i} \times F_{1,i} - \sum_{j=1}^{16} c_{7,j} \times F_{2,j} = \sum_{j=1}^{3} d_{7,j} \times F_{3,j} \end{cases} \quad (3)$$

2) Perfect Co-evolution for the Mobile Media Software Product Line: By considering the features states matrices of the mobile media and its product, the inequality (1) can be expressed as follows:

$$\begin{pmatrix} 1 & 1 \cdots & 0 & 0 \\ 1 & 1 \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \\ 1 & 1 \cdots & 1 & 1 \end{pmatrix} \times \begin{pmatrix} F_{1,1} \\ F_{1,2} \\ \vdots \\ F_{1,10} \\ \vdots \\ F_{1,19} \end{pmatrix} \geq \begin{pmatrix} 1 & 1 \cdots & 0 \\ 1 & 1 \cdots & 0 \\ \vdots & \ddots & \vdots \\ 1 & 1 \cdots & 1 \end{pmatrix} \times \begin{pmatrix} F_{2,1} \\ F_{2,2} \\ \vdots \\ F_{2,10} \\ \vdots \\ F_{2,16} \end{pmatrix}$$

TABLE V. NEW FEATURES STATES MATRIX B' OF THE MOBILE MEDIA SPL

B'	$F_{1,1}$ .. $F_{1,8}$	$F_{1,9}$ $F_{1,10}$	$F_{1,11}$	$F_{1,12}$ $F_{1,13}$	$F_{1,14}$	$F_{1,15}$ .. $F_{1,17}$	$F_{1,18}$ $F_{1,19}$	$F_{2,14}$	$F_{2,14}$
V0	1	0	0	0	0	0	0	0	0
V1	1	1	0	0	0	0	0	0	0
V2	1	1	1	0	0	0	0	0	0
V3	1	1	1	1	0	0	0	0	0
V4	1	1	1	1	1	0	0	1	0
V5	1	1	1	1	1	1	0	1	0
V6	1	1	1	1	1	1	1	1	1

We calculate the equalities (3) for the mobile media software product line and its derived product in order to deduce the results of their co-evolution:

$$\begin{cases} \sum_{i=1}^{19} b_{1,i} \times F_{1,i} - \sum_{j=1}^{16} c_{1,j} \times F_{2,j} = 0 \\ \sum_{i=1}^{19} b_{2,i} \times F_{1,i} - \sum_{j=1}^{16} c_{2,j} \times F_{2,j} = 0 \\ \sum_{i=1}^{19} b_{3,i} \times F_{1,i} - \sum_{j=1}^{16} c_{3,j} \times F_{2,j} = 0 \\ \sum_{i=1}^{19} b_{4,i} \times F_{1,i} - \sum_{j=1}^{16} c_{4,j} \times F_{2,j} = 0 \\ \sum_{i=1}^{19} b_{5,i} \times F_{1,i} - \sum_{j=1}^{16} c_{5,j} \times F_{2,j} = \underline{F_{1,14} - F_{2,14}} \\ \sum_{i=1}^{19} b_{6,i} \times F_{1,i} - \sum_{j=1}^{16} c_{6,j} \times F_{2,j} = \underline{F_{1,14} - F_{2,14} + F_{1,15} + F_{1,16}} \\ \sum_{i=1}^{19} b_{7,i} \times F_{1,i} - \sum_{j=1}^{16} c_{7,j} \times F_{2,j} = \underline{F_{1,14} - F_{2,14} + F_{1,15} + F_{1,16} + F_{1,18} + F_{1,19} - F_{2,16}} \end{cases}$$

We notice that two imperfections was detected after the calculation. They are underlined, the first in the fifth equality and the second is in the last equality. These imperfections are caused by the two features $F_{2,14}$ "Print photo" and $F_{2,16}$ "Share photo in social websites". The two features exist in the product and are absent from the software product line. The vector $A3$ is composed by the following features :($F_{1,14}$ "Add music media management", $F_{1,15}$ "Add video media management", $F_{1,16}$ "Capture videos", $F_{1,18}$ "Play videos", $F_{1,19}$ "Play music", $F_{2,14}$ "Print photo", $F_{2,16}$ "share photo in social websites"). From the seven equalities we deduce the matrix D of the inequality (1):

$$B \times A1 - C \times A2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 1 & 1 & 1 & 0 & 0 & -1 & 0 \\ 1 & 1 & 1 & 1 & 1 & -1 & -1 \end{pmatrix} \times \begin{pmatrix} F_{1,14} \\ F_{1,15} \\ F_{1,16} \\ F_{1,18} \\ F_{1,19} \\ F_{2,14} \\ F_{2,16} \end{pmatrix}$$

3) Imperfect Co-evolution Correction: In order to correct imperfections represented by the negative entries in the matrix D we propose the following algorithm which will restore the missing features to the software product line. By applying this algorithm to the mobile media software product line, the two features $F_{2,14}$ "Print photo" and $F_{2,16}$ "Share photo in social websites" of he derived product will be added to mobile media software product line states features matrix, in Table 5 we present the new matrix B' of the software product line.

Our approach allows for restoring the integrity of the software product line, by propagating features that were developed

Algorithm 1 Imperfect Co-evolution Correction Algorithm

```
//s the number of features of the SPL
//k the number of versions of the SPL
t ← s
for i ← 0 to k do
    while A3[i] ≠ A1[j] & j ≤ s do
        Imperfect ← True
        j ← j + 1
    end while
    if Imperfect = True then
        t ← t + 1
        ResizeA1(t)
        A1[t] ← A3[i]
        for r ← 0 to k do
            B[r][n] ← |D[r][n]|
        end for
    end if
end for
```

Figure 7. Imperfect co-evolution correction algorithm

on the products level to the domain engineering level. In the presented case study the two features "Print photo" and "Share photo in social websites" of the derived product were propagated to the mobile media software product line. The approach aims at preserving the software product line from the aging phenomenon by correcting the divergences between products and the software product line that happened during their evolution.

Our motivation is conducted by the main principal of software product line engineering which is the ability of the domain feature model to support all the features of the derived products. Furthermore, cladistics classification technique allowed us to restore the missing features to the corresponding versions of the software product line in order to achieve the perfect co-evolution of the software product line with its products. This approach enables also the extension of the software product line capabilities, for our example, the two added features can be propagated to the other derived products that manage photos. Thereby, it helps predicting new features and anticipating new requirements, for example the feature "Share photo in social websites" which we propagated to the mobile media software product can also be adapted to include other media such as songs and videos.

V. Conclusion

In this paper, we introduced the co-evolution of domain and application engineering in software product lines, which consists in identifying the evolution paths of the domain assets and the application assets and finding if they are similar or different. Our purpose is to preserve the ability of domain engineering assets to derive the application assets even during their evolution. This purpose can be achieved by propagating the changes that happened to the products only to the software product line. Co-evolution was extensively studied in biology in order to understand how organisms influence each other during their evolution. The co-evolution relies basically on the physical characters or behaviors of organisms. Since the features of a software represent its visible characters, we consider only the co-evolution of feature models in this paper.

We used the biological Cladistics classification to build the evolutionary trees of the software product line and its derived products, then we perform a mathematical analysis to extract similarities and differences between these trees. Thereby, we propose an algorithm to propagate the missing features that cause divergence between the evolutionary trees to the domain feature model. We applied our approach to the software product line of mobile media applications that manage media such as songs, photos and videos on mobile devices. We compared the evolution paths of the software product line and one of its derived products, then we applied our analysis to propagate the features that exists in the product but are missed from the software product line to the domain features model. As a consequence, we restored the ability of the software product line to derive all the products features, and also we extended its capabilities by the new features.

References

[1] K. Pohl, G. Bckle, and F. J. van der Linden, Software product line engineering: foundations, principles and techniques, Springer, 2005.

[2] P. Clements, L. Northrop, and B. W. Boehm, "Software product lines : practices and patterns", Fondo Xavier Clavigero, S.J. ITESO, 2002.

[3] D. L. Parnas, "Software aging", in Proc. The 16th international conference on Software engineering, 1994, pp. 279-287.

[4] P. R. Ehrlich and P. H. Raven, Butterflies and plants: a study in coevolution Evolution, JSTOR, 1964, pp. 586-608.

[5] R. M. Anderson and R. M.May, "Coevolution of hosts and parasites", Parasitology, 1982, vol. 85, no 02, pp. 411-426.

[6] D. Di Ruscio, L. Iovino, and A. Pierantonio, "What is needed for managing co-evolution in MDE?". In Proc. The 2nd International Workshop on Model Comparison in Practice, 2011, pp. 30-38.

[7] M. Kster and M. Trifu, "A case study on co-evolution of software artifacts using integrated views". In Proc. The WICSA/ECSA, 2012, pp. 124-131.

[8] C. Seidl , F. Heidenreich, and U. Amann, "Co-evolution of models and feature mapping in software product lines". In Proc. The 16th International Software Product Line Conference, 2012, pp. 76-85.

[9] Brinkman, S.L. Fiona, and D. D. Leipe, Bioinformatics: a practical guide to the analysis of genes and proteins, Vol. 43, John Wiley Sons, 2004.

[10] L. P. Tizzei, M. Dias, C. M. Rubira, A. Garcia, and J. Lee, "Components meet aspects: assessing design stability of a software product line", Information and Software Technology, Elsevier, 2011, vol. 53, no 2, pp. 121-136.

[11] N. Anquetil, U. Kulesza, R. Mitschke, A. Moreira, J. Royer, A. Rummler, and A. Sousa, "A model-driven traceability framework for software product lines", Software Systems Modeling, Springer, 2010, 9, pp. 427-451.

[12] S. A. Ajila and A. B. Kaba, "Evolution support mechanisms for software product line process", Journal of Systems and Software, Elsevier, 2008, vol. 81, no 10, pp. 1784-1801.

[13] A. Goknil, I. Kurtev, K. van den Berg, and J. Veldhuis, "Semantics of trace relations in requirements models for consistency checking and inferencing", Software Systems Modeling, Springer, 2011, 10, pp. 31-54.

 ISBN: 978-1-61208-367-4

EM^3: Software Retirement Process Model

Mira Kajko-Mattsson
School of Communication and Information Technology
KTH Royal Institute of Technology
Sweden
mekm2@kth.se

Anna Hauzenberger, Ralf Fredriksson
Department of Computer and Systems Sciences
Stockholm University
Sweden
anna-hau@fc.dsv.su.se, rafr1976@student.su.se

***Abstract*—The software community has been so much focused on creating and improving development and evolution processes, so that it has completely forgotten retirement. Today, there are no retirement process models whatsoever despite the fact that many software organizations desperately need guidelines for retiring their software systems. In this paper, we elicit theory about software retirement process and put it into a software retirement process model, which we call *EM^3: Software Retirement Process Model*. The elicitation has been done within "If…", a Nordic insurance company. The model is based on two comprehensive case studies conducted within two real-life retirement projects.**

Keywords-case studies; software lifecycle; software migration; software phaseout; software closedown; software disposal.

I. INTRODUCTION

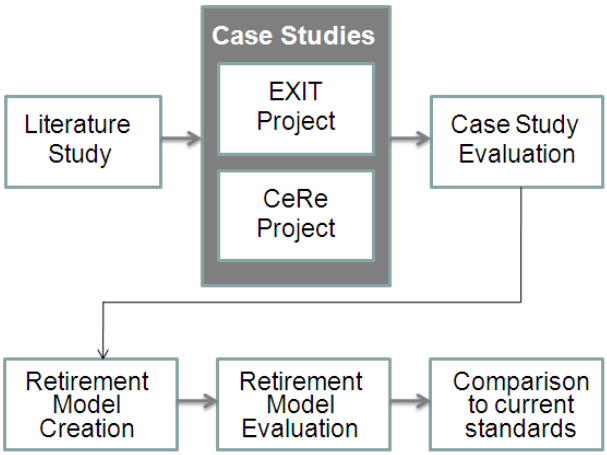
Research on software lifecycle process models has not been well balanced so far. Most of the attention has been paid to software development. Less focus has been put on software maintenance. No research been made on software retirement whatsoever.

Retirement is the disposal process whose aim is to end the existence of a software system [11]. It consists of the actual software system phaseout, removal of it from a regular usage, migration of its still relevant parts to some other system(s), closedown, and the archiving of it [1].

There are plenty of reasons why a system needs to be retired. Some of them are the system age and complexity, removal of its software and/or hardware platform, rules embodied by the external environments, and the like. Irrespective of the underlying reasons, retirement is an extremely complex and difficult process. Hence, it must be carefully planned and performed.

Right now, the concept of retirement is not well established within software engineering [5]. Neither are there any process models describing it. There are only very few standards and these standards are not based on any real-life studies [2][3]. Their contents has been mainly chosen in ballots; hence, they are very general. At its most, they cover a whole retirement process model within only a few pages. Hence, they do not provide sufficient guidelines for the organizations in their complex retirement work.

In this paper, we outline a retirement process model, called *EM^3: Software Retirement Process Model*. The model is part of *EM^3* standing for ***E**volution and **M**aintenance **M**anagement **M**odel*. The model has been created within "*If…*", one of the leading property and casualty insurance companies in the Nordic region. This company has recently undergone nine retirement projects. We have studied two of them: (1) the EXIT project performed in Sweden [8] and (2) the CeRe project performed in Finland [9]. Our goal is to provide a basis for creating theory in the domain of software retirement, to evaluate current process standards and provide feedback for their extension.



In the second case study, we explored Steps taken in our research process.

The two projects studied, differed in their prerequisites and process designs. For this reason, we made two separate case studies and put them into their respective process models [8, 9]. In this paper, we first present the two models and consolidate these them into one general model which we then evaluate within "*If…*".

The remainder of this paper is as follows. Section 2 describes our research method. Sections 3 and 4 present the EXIT and CeRe projects. Section 6 evaluates and compares the consolidated process model to the existing retirement standards. Finally, Section 7 makes conclusions and suggestions for future work

II. RESEACH METHOD

Our study was a typical design research [10]. Its goal was to explore and model the domain of retirement by identifying all its relevant process constituents and the relationships among them. As illustrated in Figure 1, our exploration work consisted of several consecutive phases.

 ISBN: 978-1-61208-367-4

TABLE I. THE EVALUATION QUESTIONNAIRE

1. Have you participated in other retirement projects, other then EXIT and CeRe?
1. What was your role within the project(s)?
3. What problems have been encountered within these projects?
4. What is your opinion about our retirement process model?
   a) Does it include the right phases?
   b) Is there any phase missing?
5. For each of the phases:
   a) Are the activities appropriate for the phase?
   b) Are there any activities missing?
   c) Do the right roles perform the activities?
   d) Can you see any problems with this phase and/or its activities?
6. How can the model solve the problems you have encountered?
7. What is good and what is bad about the model?
8. Do you have any suggestions for improvements?

In the first phase, the *Literature Study* phase, we made an extensive and comprehensive literature study. We went through various articles and standard process models touching on retirement. None of them, however, provided us with detailed information about the process. Only [2][3] outlined very general models. Due to their very coarse-grained nature, they did not provide any sufficient platform for starting our work. Hence, we may claim that our results are entirely elicited from scratch using the industrial support.

In the second and third study phases, the *Case Studies* and *Case Study Evaluation* phases, we studied both the EXIT and CeRe projects and evaluated them within the company [8][9].

Regarding the EXIT project, we studied it by first scrutinizing all the relevant project documentation. This documentation included about 100 different documents describing the retirement project, project plans, status reports, activity lists, system overviews, reports from various meetings such as steering groups, reference groups, and the like.

In the study of the CeRe project, our first step was to interview the CeRe project leader who presented the overall retirement process to us. We then continued to scrutinize relevant project documentation. This documentation included about 30 various documents.

Due to the fact that CeRe was a Finnish project, all the documentation was written in either English or Finnish. The documents written in Finnish were translated to us to Swedish by the CeRe project leader, either orally or in written.

In both the EXIT and CeRe projects, the documents studied did not fully describe the whole retirement project. Hence, we had to complement our explorative study with a series of interviews with the project leader and one operation manager.

Based on the understanding gained, we created two preliminary retirement process models for each of the retirement project studied [8, 9]. These models outlined a set of process activities in the EXIT and CeRe projects, structured these activities into process phases and identified roles involved in them. They were then presented to the project managers. The goal was to evaluate their credibility and adherence to the EXIT and CeRe projects, respectively. The evaluation step resulted in some minor modifications to the process models. These modifications are presented in Section 6.

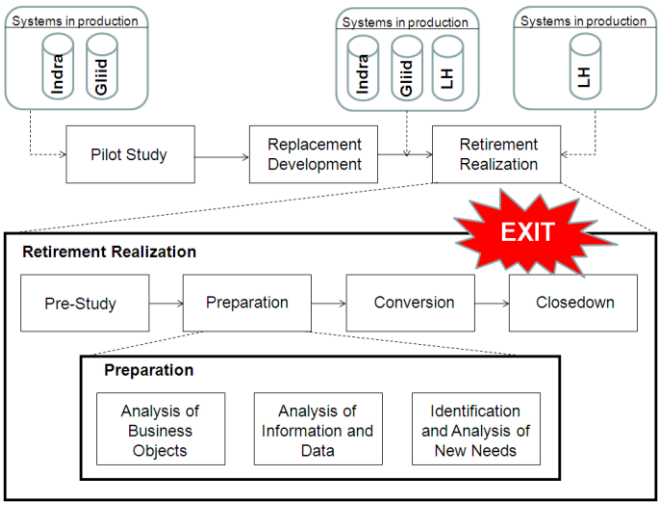


Figure 1. Phases in the EXIT project.

The process models of the EXIT and CeRe projects covered various aspects of retirement. Hence, they differed in their prerequisites and designs. As a next step, we consolidated them into one general process model, which we call *EM³: Software Retirement Process Model*. We then evaluated the *EM³* model within the company using tête-à-tête interviews. The questionnaire used for the evaluation purpose is presented in Table I.

Six people were involved in evaluating our retirement process model. Two of them were retirement project managers, one decision maker, one system analyst and maintainer, one developer and one business manager. These people were involved in at least one retirement project.

Finally, we compared our model to the standard models [2][3]. To enable the comparison, we created a set of comparison criteria. These criteria are listed in Table V. Due to the fact that the standard process models studied are very general, we could only define our comparison criteria on a very general level.

III. THE EXIT PROJECT, CASE STUDY 1

In this section, we present the EXIT project. We first present its context in Section 3.1. We then describe the project in Section 3.2.

A. Context of the EXIT Project

Two legacy systems, *Indra* and *Gliid*, were going to be retired and replaced with a system called *LH*. As illustrated in Figure 2, the overall retirement process consisted of three phases. These were (1) *Pilot Study*, (2) *Replacement Implementation*, and (3) *Retirement Realization*.

In the first phase lasting for one year, *"If..."* made a pilot study during which they examined *Indra* and *Gliid* and decided that a replacement system, *LH*, would be developed and *Indra* and *Gliid* would be retired.

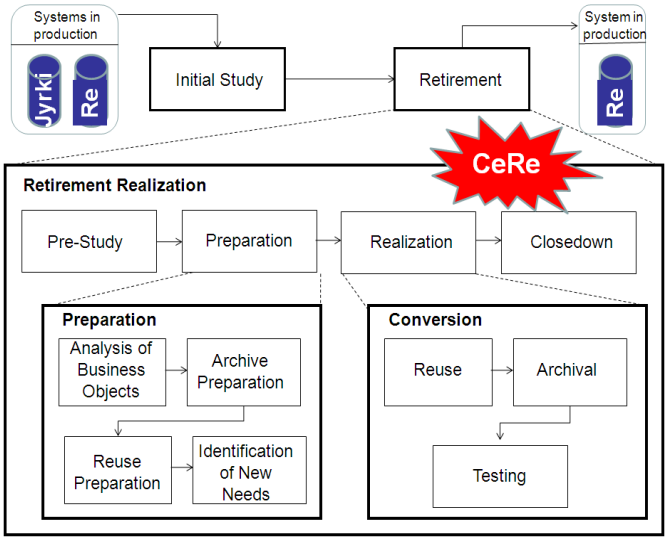


Figure 2. Phases in the CeRe project.

During the second phase, lasting for two years, "*If...*" was in the process of developing *LH*. In the next-coming two years, it started the retirement of *Indra* and *Gliid*. In the fifth year, both *Indra* and *Gliid* were closed down and only *LH* has been used since then. The star banner in Figure 2 marks the scope of the EXIT project.

B. *The EXIT Project*

The EXIT project consisted of four phases. They are (1) *Pre-Study* (2) *Preparation*, (3) *Conversion*, and (4) *Closedown*. Below, we briefly describe them.

1) Pre-Study: The goal of the *Pre-study* phase was to investigate the systems to be retired, determine which of their parts should be migrated and disposed off, identify appropriate archiving and migration strategies, define a retirement project and plan for it.

In this phase, one first investigated the types and volume of business objects to be retired and migrated. One then determined the archiving and migration needs to be further used for identifying the appropriate migration and archiving strategies. As a next step, one determined the project scope. When doing it, one first analyzed *Indra* and *Gliid*'s overall architecture and design and then identified dependencies to other interfacing systems. Here, one considered other systems and their users that were dependent on the retiring systems.

Identification of the interfacing systems affected by the closure of *Indra* and *Gliid* led to the identification of the additional activities required for managing the retirement project. In our case, one recognized a need (1) for analyzing the migration and archiving strategies, and (2) for making deeper analysis of adjacent systems and their connections to the systems to be retired.

Finally, one defined a retirement project. The project definition included risk management and creation of a retirement plan. Risk management concerned risks such as access to resources required, staff illness and various technical risks [13]. The retirement plan, on the other hand, covered most of the rudimentary project planning activities.

2) Preparation: The goal of the *Preparation* phase was to further analyze the systems to be retired, make a decision on archiving and migration strategies, determine changes to be made in the adjacent systems and in the replacing system.

As a first step, one studied the business objects to be migrated. The goal was to identify active objects and to attend to the inconsistencies in them. An example of an active business object is a car insurance.

For all the active business objects, one analyzed their individual data fields in order to determine whether they should be migrated to the new system. One also analyzed special cases. An example of a special case is when one and the same business object is administered by both systems, namely, the retiring and the replacing systems.

For the data fields to be migrated, one created a conversion table and a conversion testing plan. Testing implied that one chose a specific numeric field, summed it for all the business object instances to be migrated and compared their sum to the corresponding sum in the new system.

3) Conversion: As a first step in the Conversion phase, one developed the automatic conversion method including scripts and the automation process. This method was then tested. The purpose was to estimate conversion time and to assure a problem free conversion. When the tests were successfully passed, one conducted both the automatic and manual conversion. The conversion results were finally tested to verify that the conversion was successful.

4) Closedown: Finally, in the *Closedown* phase, one closed down the *Indra* and *Gliid* systems and removed their dependencies to the adjacent systems.

IV. THE CERE PROJECT, CASE STUDY 2

In this section, we present the CeRe project. In Section 4.1, we first present its context. In Section 4.2, we describe the CeRe project itself.

A. *Context*

In the second case study, we explored the process of retiring a system called Jyrki. Jyrki was internally developed to be used for managing debts and credits. It had about 35 users. At the beginning of the CeRe project, it was 18 years old. Together with eight other systems, it was installed on HP 3000. In the first year, HP announced that HP 3000 would be phased-out in the future five years. For this reason, "*If...*" decided to retire all the eight systems installed on this platform.

Out of the eight systems, we focused our study on Jyrki due to the following reasons: (1) it was the last system on the HP 3000 platform to retire; hence, the project results were fairly fresh, (2) having it as the last retirement project, "*If...*" had matured with respect to its retirement management; hence, the project provided us with feedback on a matured retirement process; (3) almost all the project documentation

 ISBN: 978-1-61208-367-4

TABLE II. RETIREMENT CHECKLIST

1. Identify the business criticality of the system and the need to use it.
2. Analyze system structure and its size.
3. Identify system users and their needs.
4. Estimate the system lifecycle lenght.
5. Identify adjacent systems and assess impact on them.
6. Identify all contracts applying to the retiring system.
7. Identify other systems possessing similar functionality and/or overlapping functionality.
8. Identify and study candidate environemtns to/in which the retiring system might be migrated/reused.
9. Identify the laws and rules to be considered.
10. Identify the need to reuse the system and/or its parts.
    1. Identify the business objects/functionality/data to be reused.
    2. Determine their volume.
11. Identify the need to destroy the system and/or its parts.
    1. Identify the business objects/functionality/data to be removed.
    2. Determine their volume.
12. Identify the need to archive the system and/or its parts.
    1. Identify the business objects/functionality/data to be archived.
    2. Determine their volume.
    3. Determine the archive's accessability.
    4. Determine the archive's size.

was in English; hence, we could easily follow it, and (4) many of the people involved in the retirement of Jyrki were still available; hence, they could help us in this study.

In the second year, "*If…"* decided that relevant parts of Jyrki would be migrated to an existing system called RE. RE was a standard bought-in system installed on another platform. The retirement work itself lasted for exactly one year after the decision was made to retire it.

B. The CeRe Retirement project

As illustrated in Figure 3, the CeRe project consisted of two main phases. They are (1) *Initial Study* and (2) *Retirement.*

1) Initial Study: The Initial Study phase took place in the second year. Here, one made an inventory of all the systems installed on HP 3000 and evaluated each of them. Using the checklist presented in Table II, one identified each system's criticality, analyzed its structure, users, contracts, and adjacent systems. One then studied the laws and rules to be obeyed in the process.

The analysis conducted in this phase was very general. Its goal was to provide a basis for planning future retirement work and for determining the order of retiring the systems. Later on, this analysis would be repeated for each of the retiring system.

2) Retirement

Regarding the second phase, the *Retirement* phase, it consisted of four sub-phases: (1) *Pre-Study* (2) *Preparation*, (3) *Realization*, and (4) *Closedown*.

a) Pre-Study: The goal of the *Pre-study* phase was to investigate the system to be retired, to determine which of its parts should be migrated, disposed off and archived, to identify appropriate archiving and reuse strategies, to define a retirement project and plan for it.

When investigating the CeRe project, one used the same checklist as in the *Initial Study* phase (see Table II). The goal was to find out whether its results were still relevant. This investigation was then complemented with an additional study, this time focused on archiving problems. It was led by a series of questions that are listed in Table III.

TABLE III. QUESTIONS DEALING WITH ARCHIVING PROBLEMS

1. How critical is the retiring system to the business?
2. How many users does the retiring system still have?
3. Are there any needs to migrate the retiring system's data to other systems?
4. What types of business objects are managed by the retiring system now?
5. Who generates reports and statistics in the organization and for what purpose?
6. Should the statistics and reports contain historical information?
7. How far back in time should the historical accounts stretch themselves?
8. What coupling does the retiring system have to the adjacent systems?
9. Does the retiring system need an archive?
10. What laws and requirements generate the need for an archive?
11. What are the accessibility requirements of an archive?
12. How does the retiring system influence the adjacent systems?

The *Pre-study* phase resulted in an updated plan of the continued work. The plan covered (1) the specification of the roles and activities required for conducting the work, (2) specification of the business objects to be considered, and (3) the identification of the overall strategies required for reusing and archiving the business objects. In addition to the basic strategic issues, the reuse strategy focused on confirming that RE still constituted an appropriate platform for migrating some parts from *Jyrki*. The archiving strategy, on the other hand, focused on designating the technical solution of the future archive. It was decided that Microsoft Access would be used.

b) Preparation: The *Preparation* phase encompassed a number of analyses on various levels, from business objects down to the data field level. The goal was to determine which business objects should be reused and archived and to determine the migration impact on the RE system.

As illustrated in Figure 3, the *Preparation* phase consisted of four sub-phases: (1) *Analysis of Business Objects*, (2) *Archive Preparation*, (3) *Reuse Preparation*, and (4) *Identification of New Needs.*

In the *Analysis of Business Objects* phase, one analyzed which of the business objects should be reused and archived. Here, one decided that only active objects, such as unpaid invoices, were to be reused. The objects needed for future retrieval should be archived. The rest should be disposed off. One also decided that all the reused instances should be easily traceable both in the archive and in the *RE* system. With this, one expected to have control over the migrated business objects.

For each type of a business object, one then analyzed its instances to make sure that the right ones got migrated to the *RE* system. Here, one generated lists of all the active business object instances. One then analyzed them to confirm that they had the right status. Finally, one flagged all the reused instances to make them traceable.

In the *Archive Preparation* phase, one specified requirements on the archive, identified business objects to be archived and procedures for migrating data to the archive. As a first step, one analyzed the objects on a data field level to determine which of the fields should be archived and how they should be retrieved.

 ISBN: 978-1-61208-367-4

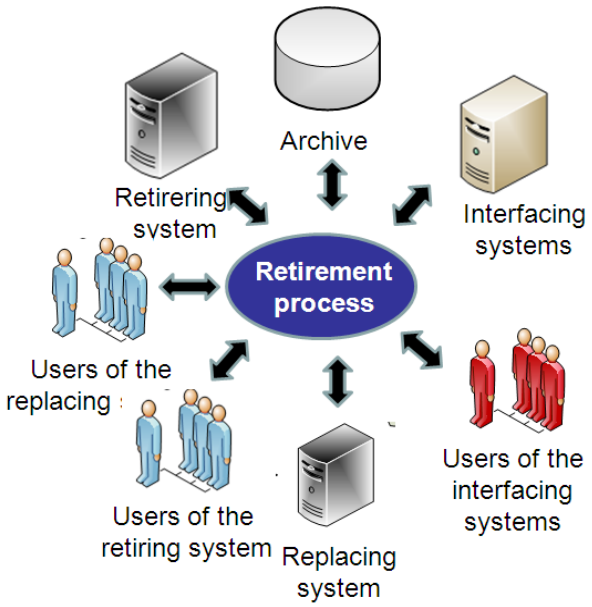


Figure 3. Components in *EM³: Software Retirement Process Model*.

As a next step, one developed a simple archive prototype. The purpose was to verify that the final archive would fulfill the organizational requirements. One then tested it and solved all the problems encountered in it. Finally, one made a decision on how to test the final archive after it got implemented.

The goal with the *Reuse Preparation* phase was to further detail the reuse strategy, specify the conversion process, revise the conversion requirements, determine testing procedures, and the like. As a first step, one established which types of business objects should be reused. For each type, one analyzed its individual fields and decided on whether they should be reused or not.

It is not easy however to reuse data fields in another system. The declarations may substantially vary. To ensure the quality of the reuse, one mapped the data fields in *Jyrki* to the data fields in *RE*. For each of them, one then determined a conversion approach, either manual or automatic.

As a next step, one specified the order in which the business objects should be converted. The order was influenced by the dependencies among the objects. For instance, customer objects should be converted first before converting their insurances. One then determined the conversion testing method. The method implied that one chose a specific numeric field, summed it for all the instances in Jyrki and compared their sums to the corresponding sum in RE.

In the *Identification of New Needs* phase, one studied whether the data migrated from *Jyrki* would affect *RE*. For this purpose, one investigated whether the working routines would have to be changed. This investigation resulted in the identification of new requests for changes to be made in *RE*. These changes were then implemented and tested. Finally, one updated the *Retirement Plan*.

3) Realization: The Realization phase consisted of the following sub-phases: *Reuse*, *Archival*, and *Testing*.

In the *Reuse* sub-phase, one first defined a conversion method. One did it for both the manual and automatic conversions. The goal was to secure that the conversion would be conducted in the right order and that nothing would be forgotten.

Regarding the manual conversion, one created a crib supporting the manual work. For the automatic conversion, one created a list identifying the automatic procedures, specifying the data to be converted and their order.

As a next step, one implemented the automatic procedures. Due to the fact that the *RE* system already implemented the automatic conversion procedure, one did not need to implement it. What one only needed was to implement procedures accessing *Jyrki's* data.

One then implemented and tested the conversion method. When testing the manual method, one converted some instances following instructions as specified in the crib. One then controlled the results. Possible problems in the manual conversion procedures were then attended to and tested anew.

When testing the automatic conversion method, one downloaded the data into the *RE*'s testing environment. One then verified the results. In case of problems, one solved them and tested the automatic procedures anew. Before starting the conversion, however, one made sure that all the preparations had been made correctly. For instance, one checked whether all the required changes had been done to *RE*. Finally, one migrated (converted) data to the RE system. One conducted the manual conversion first. Both conversions were then tested and approved.

The goal of the *Archival* phase was to create an archive, migrate data to it and test. Using the stated requirements, one started the development of the archive. One then developed the automatic procedures to transfer data from *Jyrki* to the archive. To be able to present the data in the archive, one needed reports. About ten reports corresponding to the most frequent searches in *Jyrki* were developed.

The migration of data to the new archive was tested using a sample data first. While doing it, one created a user manual and educational material. One then educated and trained its users. Finally, one conducted the entire migration to the archive. The migration was entirely automatic.

After the migration was fulfilled, one tested its results in the last *Testing* sub-phase. One did it to secure the migration correctness by comparing the data in *Jyrki* and the new archive, using similar tests as in the *Reuse* sub-phase.

a) Close down: Before one conducted the final conversion, one removed the opportunities to update *Jyrki*. However, one waited for two months before disposing off *Jyrki* and its hardware platform. This time period was a security measure during which the users could attend to the inconsistencies observed in *Jyrki*, *RE*, and the archive.

 ISBN: 978-1-61208-367-4

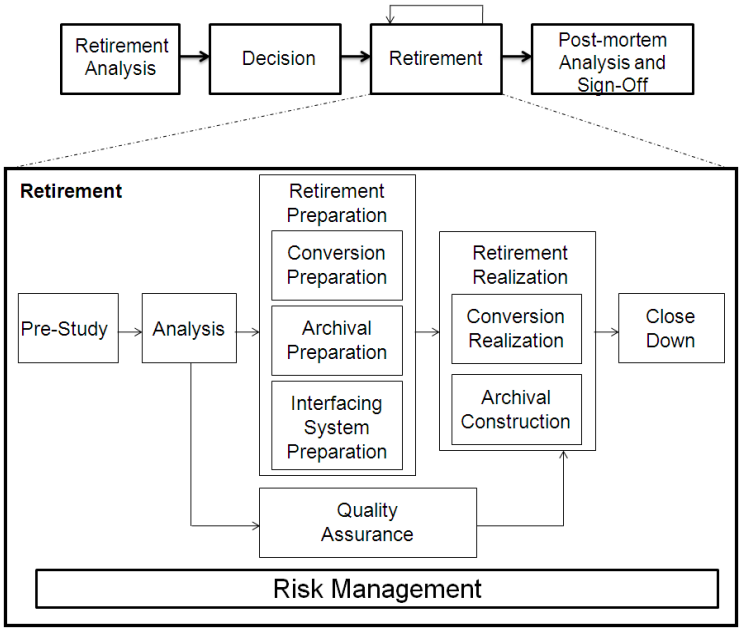


Figure 4. Design of EM^3: Software Retirement Process Model.

V. RETIREMENT PROCESS MODEL

In this section, we outline *EM^3: Software Retirement Process Model*. We first provide its overview in Section V.A. We then describe the retirement phases and roles involved in them in Sections V.A and V.B, respectively.

A. Process Model Overview

As illustrated in Figure 4, *EM^3: Software Retirement Process Model* manages components such as (1) *Retiring system(s)*, (2) *Replacing system(s)*, (3) *Interfacing system(s)*, (4) *Users of the retiring system(s)*, (5) *Users of the replacing system(s)*, (6) *Users of the interfacing system(s)*, and (7) *Archive*. Our suggestion for a retirement process model is depicted in Figure 5. It consists of four main phases:

1. *Retirement Analysis*: In this phase, one analyzes the retiring system using either the checklist presented in Table II or a decision matrix [4]. This activity is usually initiated due to many reasons. Some of them are (1) high maintenance cost, (2) removal of the software or hardware platform of the retiring system, (3) duplicated functionality in several systems [1].
2. *Decision*: In this phase, one decides whether the system should continue to provide service or whether it should be disposed off.
3. *Retirement*: If the decision has been made that the system is to be retired, then the system undergoes a retirement process.
4. *Post-mortem Analysis and Sign-Off*: After the retirement has been realized, one analyzes the process, assures that all the planned activities have been performed as planned and that all the goals have been achieved, one collects lessons learned, and finally, one signs off the retirement process.

B. Retirement Process Roles

The *EM^3: Software Retirement Process Model* retirement process involves the following roles:

- *Decision Maker (DM):* set of managerial roles responsible for planning and managing the retirement process.
- *Maintenance Organization (MO):* organization responsible for maintaining the archive.
- *Operations Expert (OE):* role possessing expert knowledge of the system to be retired and the retirement process to be conducted.
- *System Manager (SM): role* responsible for the operation and maintenance of the system.
- *System Analyst (SA):* a role responsible for planning and analyzing the system to be retired.
- *Project Leader (PL):* role responsible for the retirement project.
- *System Architect (SAR):* role is responsible for knowing the overall architecture of the systems to be retired. This is a new role added to our model after the industrial evaluation step.
- *User (U):* role using the system to be retired.
- *Developer (D):* role involved in the implementation of the retirement process.
- *Support Technician (ST):* role responsible for operation and support of the system to be retired.

C. Retirement Process Phases

The retirement process consists of five phases (1) *Pre-Study*, (2) *Analysis*, (3) *Retirement Preparation*, (4) *Retirement Realization*, and (5) *Close down*. Below, we describe each of them. As can be seen in Table IV, the total

TABLE IV. PHASES AND ACTIVITIES IN *EM³: SOFTWARE RETIREMENT PROCESS MODEL*. THE UNDERLINED ACTIVITIES WRITTEN IN BOLD WERE ADDED AFTER THE MODEL EVALUATON. TE ABBREVIATIONS IN THE PARANTHESES IDENTIFY THE ROLES PERFORMING THEM

Pre-Study

1. Analyze the system to be retired (SA, OE)
2. Study the analysis results (PL, DM, OE)
3. Identify the dependencies between the retiring system and its environment (interfacing systems)
4. Create a retirement plan (OE, SA)
   1. Identify/determine stakeholders to be involved in the retirement project (PL, DM)
   2. Identify/determine roles required for managing and executing the retirement process (PL)
   3. Determine the competence required for managing the retirement (DM, OE)
4. Plan for how archiving, reuse and closure (SA, OE)
5. **Manage risks**
   1. **Identify risks (PL, OE, DM, SA)**
   2. **Analyze risks (PL, OE, DM, SA)**
   3. **Make a decision on how to manage risks (PL, OE, DM, SA)**
6. Create a retirement stratey (DM, OE, SA)
7. Create an archiving strategy (DM, OE, SA)
8. Determine mistones and dates (DM, PL)
9. **Determine budget (PL, DM)**
10. ….

Analysis

1. Analyze the business objects to be migrated/archived. For each type:
   1. Identify the business objects to be migrated/archived (U)
   2. Identify the business objects to be quality assured (DM, OE, SA)
2. Analyze the business objects to be migrated
   1. Identify special cases of business objects (OE, SA)
   2. For each type of business object
      1. Study existing conversion techniques (automatic/manual) (DM, OE, SA)
      2. Analyze its individual data fields in the retiring system (OE, SA)
      3. Analyze its individual data fields in the replacing system (OE, SA)
      4. **Create conversion rules (OE, SA)**
      5. Create a conversion table between the retiring and replacing system (OE, SA)
      6. Create a conversion testing plan (OE)
      7. **Determine the order of converting the business objects (OE, SA)**
5. Analyze the business objects to be archived
   1. **Identify special cases of business objects (OE, SA)**
   2. **Study the archiving method (SA, D, MO)**
3. For each type of business object
   1. Analyze its individual data fields in the retiring system (OE, SA)
   2. Analyze its individual data fields in the archive (OE, SA)
   3. **Create conversion rules (OE, SA)**
   4. Create a conversion table between the retiring system and the archive (OE, SA)
   5. Create a conversion testing plan (OE)
   6. Analyze how the information is to be presented in the archive (OE, SA)
   7. **Determine the terminology to be used in the archive (OE)**
4. Develop the archive's prototype (SA, D)
5. Test the prototype (OE)
6. **Determine how the business object which are going to be neither migrated nor archived should be managed (DM, OE)**
7. Modify the dates in the plan, if needed (DM, OE, MO)
   1. Date when the retiring system is to be closed for update
   2. Date when the business objects should be migrated to the replacing system
   3. Date when the retiring system system should be retired

Quality Assurance

1. Analyze the business objects and their instances.
   1. For each type:
      1. Determine how the business objects should be analyzed (OE, SA)
1. Analyze the instances of the business objects
   1. Assure that the instances are flagged in the archive (OE)
   2. Make necessary changed to the instances, i.e., update their status (U)

Retirement Preparation

1. Prepare for the conversion
   1. Develop conversion process/procedures
      1. Describe how the conversion will be performed (both the manual and automatic conversion) (D, OE)
      2. Develop the extraction and loading processes/procedures for the automatic conversion (D)
   2. Test
      1. Test the manual conversion (OE)
      2. Test the extraction and loading processes (D)
      3. **Conduct functionality tests in the replacing system using the converted data (OE)**
2. Prepare for the archiving activity
   1. Develop the archive and the archiving process
      1. Develop the automatic procedures/procedures for accessing and storying data (D)
      2. Adapt the archive to the rules specified (D)
   2. **Determine the maintenance organization which will take over the archive (DM)**
   3. Test (D, OE)
      1. Test the automatic procedures
      2. Test the archive
   4. Create a user manual (OE)
   5. Educate and train
      1. Create educational material (OE)
      2. Conduct the education and training (OE, U)
3. Prepare the environment (interfacing systems)
   1. Analyze dependencies between the retiring system and the environment (SF, SA)
   2. Analyze the manual working routines (OE, SA)
   3. Analyze the consequences in the replacing system and the interfacing systems (OE, SA)
   4. Issue change requests and make changes in the replacing and interfacing systems, if required (DM, OE)
   5. Test the changes (DM, OE).

Retirement Realization

1. Perform conversion
   1. Conduct the manual conversion (U)
   2. Conduct extraction and loading processes (D)
   3. Make sample tests in the retiring and the replacing systems (U, OE)
2. Perform archiving
   1. Conduct the archiving work (D)
   2. Make sample tests in the retiring system and the archiOE (U, OE)
   3. **Transfer the archive to the maintenance organization (PL)**

Close Down

1. Remove the opportunity to use the retiring system (MO)
2. Close down and remove the automatic procedures (MO, DT)
3. Assure that all the planned activities have been conducted (PL, OE, DM)
4. Dispose off the retiring system (MO, DT)

 ISBN: 978-1-61208-367-4

process is structured into phases and activities. The goal is to create a reference framework mapping out what activities are relevant in what phase. However, the order of the activities as listed in our model does not impose any specific order of conducting them. Depending on the context at hand, these activities may or may not be selected. If selected, then they may be implemented in the order that is suitable for the context at hand.

5.2.1. Pre-Study. The *Pre-Study* phase starts only after one has made a decision that the system of concern is going to be retired. Here, one makes a comprehensive and detailed analysis of the retiring system. When doing it, one may use a checklist as presented in Table II, the same checklist that has been used in the more general *Retirement Analysis* phases. The goal is to get an understanding of the retiring system and to create an overall retirement plan.

5.2.2. Analysis. In this phase, one performs deeper analysis of the retiring system in order to get an understanding of the forthcoming retirement process. Here, one identifies the business objects to be managed and their underlying functionality. One then decides how they should be handled and one designates retirement project dates. One also decides on the quality levels for securing the management of the business objects.

5.2.3. Quality Assurance. This phase starts after one has determined which business objects should be quality assured. It runs in parallel with *Retirement Preparation* and partly with *Retirement Realization*. Here, one determines the rules for quality assurance and conducts the quality assurance.

5.2.4. Retirement Preparation. In this phase, one (1) prepares the system parts to be reused in the replacing system, (2) one prepares the system parts to be archived, and (3) one studies the impact of the conversion and retirement processes on the interfacing systems.

5.2.5. Retirement Realization. In this phase, one conducts the actual conversion and archival of the business objects and their underlying functionality.

5.2.6. Close Down. In this phase, the retiring system gets closed down and disposed off. Its data may be accessed only in its archive.

VI. EVALUATION RESULTS

This section presents the results of evaluating our retirement model. Section 6.1 first presents the evaluation results of the EXIT and CeRe projects. Section 6.2 describes the evaluation results of our model.

A. Evaluation of the EXIT and CeRe Projects

In the third phase of our study (see *Case Study Evaluation* in Figure 2), the models of the EXIT and CeRe processes were presented to the project managers responsible for the respective retirement project. According to them, our models were realistic and they fully reflected their retirement processes. They had, however, some minor deficiencies. These concerned lack of three important activities: *Activity 5 (Manage Risks)*, *Activity 9 (Determine budget)* in the *Pre-Study* phase, and *Activity 2.2.7 (Determine the order of converting the business objects)* in the *Analysis* phase. They also concerned lack of the role of *System Architect*.

According to our interviewees, risk management constitutes an essential activity within retirement. Not doing it implies a critical business risk by itself. Risk management should be run continuously throughout the whole retirement project. Due to the difficulties of integrating its activities with our retirement model, we only mark their start in the *Pre-Study* phase. However, in Figure 5, we place risk management as a parallel phase to the entire *Retirement* phase.

Regarding the second activity, the activity concerning the determination of retirement project budget, our interviewee from the EXIT project claimed that due to the project criticality, it is very important to assign substantial resources to the retirement project. Otherwise, one runs the risk of underestimating the project scope and thereby fails with its completion.

We admit that this planning activity is very important. When creating process models of the individual processes, we were mainly focused on identifying pure retirement activities. On purpose, we left out many activities typical of a traditional project planning. To remedy this, we have expressed the need for more project planning activities with three dots in *Activity 10*.

The third activity, *Activity 2.2.7 (Determine the order of converting the business objects)* in the *Analysis* phase, concerned the specification of the order of converting business objects. Some objects, should be converted first before converting the other objects. For instance, client objects should be converted before their insurance objects. As a response, we have added this activity to our model.

One role was claimed to be missing within the first evaluation phase. It concerned *System Architect*. According to both project managers, this role is indispensible in all the retirement projects. Not only does this role know the system to be retired but also all its architectural flaws and deficiencies that should not be migrated to the new system.

B. Evaluation of EM^3: Software Retirement Process Model

In the fifth phase of our research (see *Retirement Model Evaluation* in Figure 2), the model was presented to six software professionals within *"If..."*. As already mentioned in Section 2, all of them were involved in at least one retirement project.

According to the "If…"'s software professionals, our retirement process model is realistic and appropriately mirrors the retirement process. They have however had some comments and suggestions for its improvement. Some of them have been attended to by complementing the model with additional activities. Those which could not be attended to immediately constitute our suggestions for future work.

The activities that have been added are:

- *Activity 6* in the *Analysis* phase (*Determine how the business objects which are going to be neither migrated nor archived should be managed*). It is important to analyze and make decisions on all the objects in the

TABLE V. OUR COMPARISON RESULTS

Activities	IEEE Std. 10741991	ISO/IEC 15288	EM³: Sw Retirement
- System Analysis	—	—	+
- Archiving Strategy	—	+	+
- Migration Strategy	—	—	+
- Mngt of Adjacent Systems	—	+	+
- Retirement Planning	+	+	+
- Risk Management	—	—	+
- Conduct Archival	+	+	+
- Designate Maintenance Org.	—	—	+

retiring system. It is only then one may make sure that one has not omitted any business object.

- *Activity 2* in the *Retirement Preparation* phase *(Determine the maintenance organization which will take over the archive) and Activity 2.3* in the *Retirement Realization* phase *(Transfer the archive to the maintenance organization):* These two activities are very important. Our interviewees claim that the maintenance organization should be designated as soon as possible and that it should play the driving role within the retirement project.
- *Activity 3* in the *Close Down* phase *(Assure that all the planned activities have been conducted):* According to our interviewees, one needs an additional activity in the *Close Down* phase to assure that all the planned activities have been successfully implemented.

During the interviews, one issue was raised. It concerned information dissemination and documentation. Information dissemination has been regarded as a very important process activity. If not properly performed, it may lead to many problems. Regarding documentation, it is important that it is pervasive throughout the whole process. Our interviewees claim that all the process phases and activities should be thoroughly documented to assure that retirement gets implemented in a proper way. It is especially important that the conversion and archiving processes, and archive manuals are documented.

Our interviewees have also identified some problems within their respective retirement projects. The problems are:

- *Too little effort has been put into the analysis of the retiring system:* This has prolonged the retirement process due to the fact that additional work was required for repeating the analysis steps.
- *Lack of resources:* It is difficult to estimate the resources required for retiring the system. This is due to lack of retiring experience and too little effort put into the analysis activities.
- *Difficulties to man the retirement projects:* It is difficult to find individuals possessing the right competence for retiring software systems.
- *Retirement projects are not high priority projects:* Retirement projects are less prioritized than other projects. This in turn prolongs their duration.
- *Weak decision making*: Retirement is a very complex activity during which many important decisions are taken. They concern decisions whether to migrate, archive, or dispose off. Hence, key individuals must be assigned clear responsibilities to make decisions. They must also be members in the project and display interest and engagement in the retirement work. Lack of it may lead to the overall project delay.

Finally, our interviewees made a suggestion that one should wait with the physical disposal of the system for a while. In this way, one makes sure that no important activity or decision has been missed. If, for some reason, defects have been injected, one may still attend to them before it is too late.

C. Comparison to Standards

In this section, we compare our retirement process model with the standard process models as described in IEEE STD 10741991 [2] and ISO/IEC 15288 [3]. When doing this, we follow the comparison criteria listed in Table V. Except for the criteria concerning the roles, all the comparison results are listed in Table V.

None of the standard process models suggests any roles to be involved in the retirement process. Only the *IEEE* model mentions a user role, who should be notified about the closure of the system. Our model however has identified ten different roles. These are listed and described in Section 5.2.

The broad portfolio of the roles identified in our model indicates that the retirement project involves the majority of the organizational roles ranging from user to various analyst and design roles, to managerial roles and even to front-end support roles [6]. This, in turn, indicates how complex and comprehensive the retirement process model is.

As illustrated in Table V, none of the standard process models include the activities during which one analyzes the retiring and the replacing systems. In accordance with the opinion of our interviewees, we believe that these are one of the most important activities within the retirement process. They could be compared to the requirements specification activities. It is a common knowledge that a non-recognition of the requirements, irrespective of what type of a project it concerns, does not lead to successful project results. For this reason, we claim that lack of analysis activities is a series deficiency in the standard process models studied.

Only the *ISO/IEC 15288* standard suggests identification of archiving strategies. None of the standards proposes migration strategy. In our opinion, identification of both these strategies is very important. Identification of the retirement strategy is a must. However, the identification of the migration strategy should be an option. This is due to the fact that not all retiring systems undergo migration. We believe, however, that the inclusion of this strategy in the retirement process model indicates that the retirement process does not exist in a vacuum. Many times, parts of the retiring systems have to be migrated to other new replacing systems or other new archiving systems.

Only the *ISO 15288* standard briefly mentions that the interfaces to the adjacent systems should be considered. None of the standard models suggests how the interfacing systems and their users should be handled. In our opinion,

 ISBN: 978-1-61208-367-4

this is a serious omission. Improper management of the adjacent systems may lead to big inconsistencies and problems in their future operation. Hence, we suggest that the interfacing systems and their handling should be highly prioritized in a retirement process.

Both the standard process models studied include the planning activities. However, they only recognize the need for planning. They have not provided any suggestions specific to the retirement planning process.

None of the standard process models studied included risk management. We did not include it either in our preliminary process model outline. Even if risk management is a separate process, we strongly believe that it definitely should be integrated with the retirement process. Retirement and replacement imply many serious business risks. Not considering them may jeopardize the whole retirement process, and thereby, the organization's future business opportunities.

All the standard process models included the archival activity. This activity however was only briefly mentioned, even in our process model. We suspect that this activity is quite complex. Hence, it should be further scrutinized in the future.

Finally, none of the standards designates the maintenance organization responsible for driving the retirement project and for maintaining the archival. We believe that including maintenance organization right from the beginning helps avoid many future maintenance problems.

VII. FINAL REMARKS

In this paper, we have outlined a retirement process model. The model is called *EM^3: Software Retirement Process Model* and it is part of *EM^3*. It has been designed and evaluated within "*If…"*, a company that has recently undergone nine retirement projects.

Except for a very few standards, there are no retirement process models whatsoever. Hence, we dare claim that our work is unique and innovative. Our results are entirely designed from scratch using the industrial support. Hence, this paper is one of the first reports on this very complex process. More work however needs to be made to both validate and elaborate on our process model. We, therefore, cordially invite the software community to help us with this very exciting project.

REFERENCES

[1] S. W. Ambler, M. J.Vizdos, and J. Nalbone, The Enterprise Unified Process :extending the Rational Unified Process Upper Saddle River, N.J. :: Prentice Hall PTR, 2005.

[2] IEEE Standard for Developing Software Life Cycle Processes, IEEE Std 10741991. 1991. The Institute of Electrical and Electronics Engineers, Inc.345 East 47th Street, New York, NY 10017-2394, USA, 1991.

[3] ISO/IEC 15288, Systems and Software Engineering – System life cycle processes, IEEE Std 15288-2008, 2002.

[4] I. Jacobson, F. Lindström, Reengineering of old systems to an object-oriented architecture. *SIGPLAN Not. 26*(11), 340–350, 1991.

[5] M. Kajko-Mattsson, "Common Concept Apparatus within Corrective Software Maintenance" International Conference on Software Maintenance, IEEE Computer Society Press: Los Alamitos, CA, Sep 1999, pp. 287-297, ISBN: 0-7695-0016-1, doi: 10.1109/ICSM.1999.792626.

[6] M. Kajko-Mattsson, L.-O.Tjerngren, A. Andersson, "CM^3: Up-front Maintenance", Conference on Software Engineering and Knowledge Engineering, Knowledge Systems Institute, 3420 Main Street, Skokie, IL, 60076, USA, 2001, pp. 371-378.

[7] M. Kajko-Mattsson, "Corrective Maintenance Maturity Model: Problem Management", PhD thesis, Department of Computer and Systems Sciences (DSV), Stockholm University and Royal Institute of Technology (KTH), 2001, ISBN Nr 91-7265-311-6, ISSN 1101-8526, ISRN SU-KTH/DSV/R--01/15.

[8] M. Kajko-Mattsson, R. Fredriksson, A. Hauzenberger, "Elicting a Retirement Process Model: Case Study 1" International Conference on Computer Science and Software Engineering, IEEE, 2008, doi: 10.1109/CSSE.2008.1364 .

[9] M. Kajko-Mattsson, R. Fredriksson, A. Hauzenberger, "Elicting a Retirement Process Model: Case Study 2", International Conference on Innovation in Software Engineering, IEEE, 2008, doi: 10.1109/CIMCA.2008.94.

[10] B. Laurel, Design Research: Methods and Perspectives, the MIT Press, 2003.

[11] V. T. Rajlich, K. H. Bennett, A staged model for the software life cycle. *Computer 33*(7) 2000.

[12] I. Sommerville, *Software Engineering* (8th ed.). Addison-Wesley, 2007.

[13] M. Kajko-Mattsson and J. Nyfjord, "State of Software Risk Management Practice", International Journal of Computer Science, IAENG, vol. 35, iss. 4, pp. 451-462, 2008.

 ISBN: 978-1-61208-367-4

Causality Control in Dynamic Platforms

Jacky Estublier, Germán Vega
Université Grenoble Alpes, LIG
Grenoble, France
{Jacky.Estublier, German.Vega}@imag.fr

***Abstract*— The increasing dynamicity of ubiquitous environments and the rapid penetration of many sensors in our day life are causes of concern for application designers and developers. Indeed, they have to implement reliable applications in a context in which the managed entities have a very low level of abstraction; they are autonomous, heterogeneous, and change in unpredictable ways. To simplify developers work, there is a clear need to define a higher level of abstraction in which these entities can be represented homogeneously and managed systematically, irrespective of the many technical details. To be used safely, this representation must be causally related to the represented entities. Providing a high level causal representation is very challenging, because its implementation depends on the nature of the managed entities, and because in ubiquitous systems the representation and the system are evolving simultaneously and independently, sometimes in incompatible ways. The paper describes a systematic and extensible way to define and implement causality, and presents the experience with the Apam system in the domain of service platforms.**

Keywords- component; model; services; platform; causality; operational; OSGi.

I. INTRODUCTION

Almost every piece of information managed by a program is a representation of something, either abstract concepts (integers, strings), or real entities (persons, cars). An important part of computing sciences has been devoted to representations. In the 2000s, modelling proposed to make more formal the relationship between a representation (a model) and the system being represented, the System Under Study (SUS).

When a part of the SUS is not directly accessible to the machine (e.g., a part of the "real world"), building a representation is a preliminary step before writing a program that works on the SUS. A fundamental property of a model is to provide a convenient representation of the SUS: it should only represent what is needed at the right level of abstraction, making the understanding easier, and making the programs simpler. Therefore, even when the SUS is itself abstract, it is often convenient to build "on top" of it, a representation that fits better the needs.

Note that the SUS itself can be a representation of a lower level system, making SUS and representation relative concepts. Indeed, computer sciences make heavy use of chains of representations, like abstraction layers in an operating system.

The intuition often makes a distinction between SUS that are part of the real world (e.g., cars and houses represented in a database), and SUS that are electronic entities (files and ports in an operating system). This intuition is often misleading, machine world and real world are not two separate worlds; after all, the machine too pertains to the real world, and the SUS can include entities pertaining to the machine.

However, what is relevant is that for electronic entities changing the representation can be translated automatically, and almost instantaneously, into corresponding changes on the represented entity (e.g., closing a port or changing the value of an integer Java variable). We say that the representation is operational. It is of course not the case for real world entities (changing the color of a car registered in a data base does not actually change the color of the car itself).

For electronic entities, system changes can be directly observed and translated into the corresponding representation. We say that the representation is sensitive to its SUS. For real world entities, it is a program, or an administrator, that keeps the representation up to date, not the entity itself.

A representation that is both operational and sensitive (as illustrated in Figure 1) is said to be causally related to its SUS, and causality is the relationship between a representation and its SUS. Operationality and sensitivity are reciprocal properties, making causality symmetric, and making relative the concepts of SUS and representation.

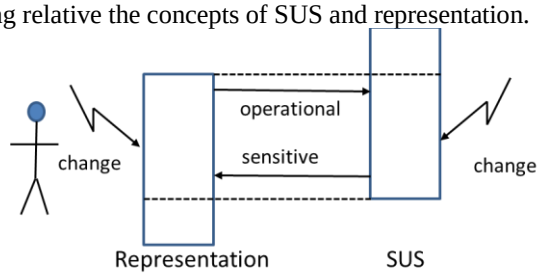


Figure 1. Causality.

With the advent of modeling as a discipline, the representation has become more formal and higher level, often based on Object-Oriented concepts, making program and representation pretty close, blurring even more the boundary between system and representation.

With the recent irruption of many sensors and actioners (ubiquitous computing, home automation, games, and so on) the machine and the real world became intertwined, because electronic devices have the property to be both in the machine world and in the real world; we call it the shared world.

 ISBN: 978-1-61208-367-4

Unfortunately, the electronic side of devices being very low level, the need for an abstract and convenient representation does not disappear (we still need to hide heterogeneity, and many technical details like communication and discovery protocols). Such a representation of the shared world can be operational, i.e., changes on the representation can be automatically translated into the corresponding actions on the associated device(s).

Conversely, devices being part of the real world, their state can be changed by the real world itself (a temperature sensor, for example). Being in the shared world, this (real) change can be translated into the corresponding change into its abstract representation, making the representation sensitive to its SUS.

Causality is transitive allowing the definition of chains of representation, of increasing abstraction, each layer being still causally related to the "lowest" one. This property is well known for operationality (the usual abstraction layers), with causality it allows, for example, to represent and manage sensor networks at the relevant abstraction level. Therefore, causality allows program to work on the representation as if working on the SUS itself, even for dynamic and autonomous SUS. This is an important property that simplifies dramatically the writing of program. However, causality is a relationship that is very difficult to enforce in practice, which explains why it is so uncommon.

We have experimented how causality can be defined and managed in a systematic way in the case where the representation is a model, close to Extended Entity-Relationship (EER), and the SUS is a software services platform (like OSGi [1]) both running on same computer. However, this simplification does not reduce significantly the generality because, in our system, everything is represented as a service: the shared world entities (sensor drivers are services), remote entities (their proxy are services), and so on.

This paper is structured as follows: Section II describes the representation layer (a component model), the execution platform (the SUS) and how causality is defined. Section III describes how the representation and the execution platform are synchronized; Sections IV and V show how this representation can be extended to handle provisioning and how it can cope with failure; finally, we conclude with a discussion of our validation and experience, the related work, and perspectives.

II. THE APAM REPRESENTATION LAYER

The Application Abstract Machine (Apam) platform proposes to its users (program and administrators) the mechanisms to build models that are causally related to their SUS. All representations in Apam conform to the meta-model depicted in Figure 2.

Apam proposes a generic Entity-Relationship meta-model (left part of the figure) that can be used to build any abstract representation, particularly for real-world entities. This generic meta-model has been specialized into a component meta-model (center of the figure) that is used specifically to represent services and running applications (the machine-world) of a service platform like OSGi.

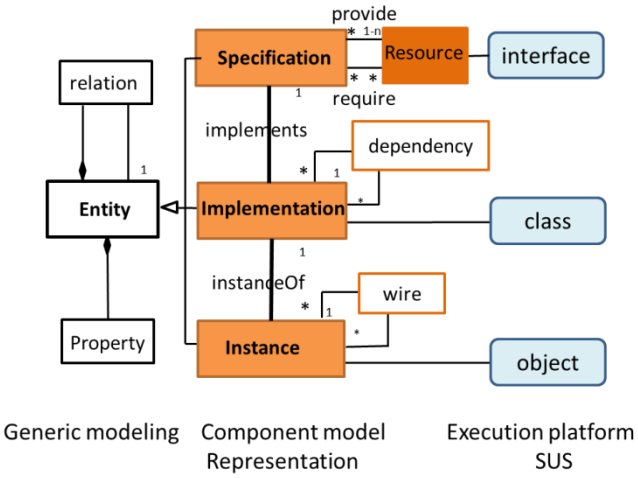


Figure 2. Apam Metamodel (simplified view).

Apam maintains a causality relationship between the abstract representation of the application and its concrete code artifacts (Java code in our case), both at development and runtime.

At development time, causality is enforced by the Apam compiler. The compiler ensures that the abstract relationships defined by the component model are actually implemented at the code level. For example, in the meta-model the relation *implements* means that the resources provided and required by the associated *specification* must be provided and required by its *implementations*; the compiler checks that the associated class really provides (implements, in the Java sense) and requires (imports, in Java code) the interfaces associated to the resources. The complete component model, and its Java mapping, is fully presented in [2]. The Apam compiler also performs byte code instrumentation to enable monitoring and management at execution.

The causal relationship established at development-time between the component representation and the actual code allows reasoning about the application completely in architectural terms. It also enables to control the execution of the application by manipulating the model at runtime, as presented in the following section.

III. CAUSALITY CONTROL

The component model and the causality control in Apam have been primarily intended to monitor (sensitivity) and drive (operationality), at high level, the execution of applications on top of a service platform. The represented SUS is the execution of application services. Those services in turn can represent the state of devices, the sensed activity or actions provided by actuators.

The service execution layer is based on the OSGi [1] and iPOJO [3] platforms. OSGi provides the basic mechanisms for deployment, live update and dynamic service discovery. iPOJO provides the component container and dependency injection mechanisms. In the execution platform, an application is, at any given point in time, a particular assembly of concrete OSGi service instances. The execution

 ISBN: 978-1-61208-367-4

platform handles the deployment and instantiation of the actual Java code of services, and service binding is automatically performed by the iPOJO container, using injected fields in the class of the client service.

At the representation level, the application is represented as a dynamic and reconfigurable architecture, composed of component instances linked by wires. The APAM platform controls the execution by continuously resolving the required dependencies and changing the model.

To effectively drive the execution of the application, this layer must be causally related to the actual service execution layer, as illustrated in Figure 3. Each change of the architecture, like creating components and wires is transformed into actions in the execution platform. For example, creating a wire from source instance *s* to target instance *t* at the architectural level produces the injection of the address of *t* into the fields of *s* in the Java code of the *s* implementation. In this regard, the Apam representation is a virtual machine executing the architectural application description, on top of the underlying OSGi execution platform.

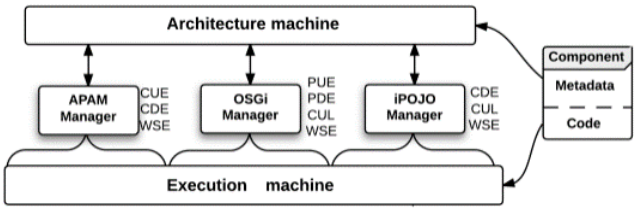


Figure 3. Causality Managers.

However, in a dynamic and ubiquitous context there are a number of external and uncontrolled events that may affect the application execution; for instance, new devices can be discovered / removed that need to be integrated / removed in the application, legacy components can be installed in the platform offering services required by the application, other applications can be installed that may interfere with the application. The execution platform automatically detects three kinds of changes:

- Components that appear and disappear
- Property changes of a component
- Service binding request from a component

The representation also manages components, properties and wires; indeed, in some cases, context changes, detected by the execution platform must be transformed into the corresponding change in the representation. For example, the apparition of a new device is important for the application architecture since it may trigger application adaptation to this new context (like making use of such a device). This requires bidirectional synchronization between the two platforms; that is the responsibility of the causality managers.

A. Causality Managers

In a top-down view of the execution, the application description presented in Section II is a specification that must be enforced in the execution platform. In a bottom-up view, the context changes detected by the execution platform must be represented in the architectural layer, in order to trigger the appropriate adaptations. There is thus a need to enforce a causal relationship between the two platforms.

Both platforms share the concept of components having properties and wires; at different levels of abstraction. For the architecture platform, a component is a description (its metadata), and wires are relationships between these descriptions; while for the execution platform, components are classes and objects, and wires are addresses into Java fields. Properties are similar in both platforms.

Causality managers are in charge of keeping the two platforms synchronized. Each causality manager is driven by a model (illustrated in the middle part of Figure 3) expressing its synchronization strategy along three axes:

1) What to change: as expressed above, the three shared concepts to synchronize are components, properties and wires (labeled C, P, W respectively in the figure).
2) Direction to change: a causal manager may be operational, propagating changes from the architectural platform to the execution platform, (labeled D for Downwards); or sensitive, propagating changes from the execution platform to the representation (labeled U for Upwards). In some cases, both platforms collaborate to take a decision; (labelled S for Symbiotic).
3) When to change: propagation can be Eager (labeled E), meaning that it happens as soon as the change occurs, or it can be Lazy (labeled L), meaning that the propagation will be done only on demand by the other platform.

For components directly specified using the APAM component model, at development time, the Apam compiler automatically includes the metadata described in Section II. The Apam causal manager extracts this metadata from the packaged component, and builds the corresponding component in the architecture platform.

For other legacy component technologies, a causal manager is in charge to extract the available information and to build the corresponding architectural object. However this requires a deep knowledge of each technology, hence a specific manager (for instance, the legacy OSGi and iPOJO managers in the figure).

B. The Apam Causal Manager

The strategy used by the native Apam component is an immediate causality for components: CUE and CDE, i.e., as soon as an Apam component appears (C for component), whether in the architecture or execution platform (Upward, and Downwards), it is immediately (Eager) synchronized on the other platform.

The code of Apam native components is injected to intercept all references to the fields of the required dependencies. The need to resolve a wire (Wire) is detected by the execution platform which decides, in symbiosis with the architecture platform (Symbiotic) to immediately (Eager) resolve the wire in both platforms, hence WSE synchronization. Properties are not synchronized since they are only known and used by the architecture platform.

 ISBN: 978-1-61208-367-4

C. Legacy Causal Managers

For the OSGi causal manager, the available information is limited to the properties published in the OSGi registry; and properties can be modified in both the architecture and execution platforms, hence the PUE, PDE synchronization: Properties are synchronized Upward and Downward Eagerly. OSGi component can be created only by third parties in the execution platform (they do not have factories), hence the CUE synchronization.

iPOJO causal manager is still another case: iPOJO factories can be used to create and instantiate components at the architecture and at the execution platform layers. Components created by the architecture platform must immediately affect execution, hence the CDE synchronization. Conversely, legacy iPOJO components are synchronized up only when required: CUL synchronization. Wires are Symbiotically, Eagerly synchronized (WSE). Properties are only visible and used in the architecture platform, and thus are not synchronized.

IV. PROVISIONING EXTENDED CAUSALITY

Thanks to the sensitivity property, in our case, the representation allows monitoring the services currently running in the execution platform and deployed by third parties, using platform specific mechanisms. The operationality property requires the capability to add/remove/create entities (components and instances) at the representation level, not only to manage those already existing in the execution platform.

To satisfy this requirement, Apam includes the capability to perform component provisioning. At the representation level, this provisioning capability is used to satisfy the dependencies of the application, when a resource is required. In practice, to find the needed component(s) and resources the Apam kernel looks into a number of search spaces.

Search spaces in turn are mapped to concrete service repositories, of diverse and open-ended nature: it may include components repositories, existing cloud services, networked devices, or even remote platforms.

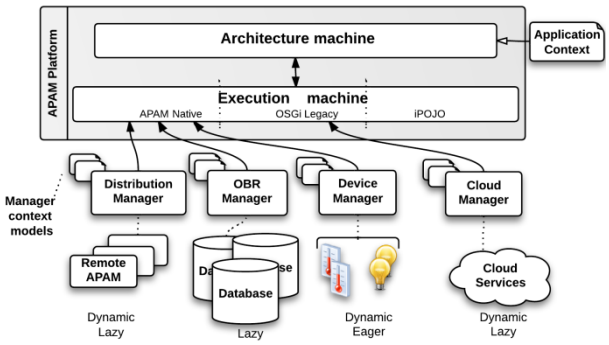


Figure 4. Provisioning managers.

Apam proposes provisioning managers as an extensible mechanism to control the search spaces. Figure 4 shows the currently defined provisioning managers, with their behavior.

A. Provisioning managers

We qualify as provisioning managers the managers in charge of synchronizing the execution machine with other platforms. We call platform any repository containing services that provisioning managers can access from the execution platform; directly by deployment, or indirectly through a proxy. Provisioning managers synchronize an "external" platform with the execution platform. We distinguish Lazy vs. Eager and Dynamic vs. Static behaviors for provisioning managers. Lazy and Eager have been already discussed; Dynamic means that changes in the external platform are "immediately" synchronized with the execution platform; Static means that changes, if any, are not synchronized.

B. Causal provisionning managers

Eager and Dynamic provisioning managers are those that define what constitutes the execution context of the application, since each change in their platform is immediately perceived by the execution platform, which, depending on its causality, the manager transfers its perception to the architecture platform. Here, the context is made of Apam and all the devices controlled by the device manager. Other context managers can be defined and added (dynamically or not) to the Apam system.

C. Lazy managers: an extended search space

The Lazy managers define the search space in the following way: when Apam tries to resolve a wire, it looks for a satisfactory target in the architecture machine. If the target is not found Apam delegates resolution to available Lazy managers, because lazy managers may know components in their platform that are not (yet) present in the architecture machine. These managers must implement the method *resolve (Dependency d, Composite context)* which returns if found an instance in the execution machine satisfying the dependency d in the provided context.

The architecture machine invokes each lazy manager in their priority order until one returns an instance *t'*. Apam invokes the relevant causality manager to reify *t'* as a *t* instance in the architecture machine, and return *t* as the resolution solution. If no manager finds a solution, the resolution fails.

Many Lazy managers can be defined, Apam provides with the standard distribution the OBR manager that can deploy components from a list of bundle repositories; the Distribution manager that looks for a component in another remote Apam machine and returns a proxy towards the selected remote component. The Cloud manager, based on the Rose framework [4], returns a proxy toward a remote service (WS, etc.). Other managers can be defined; they will be called if registered as a dependency manager.

V. FAILURE HANDLING

Even with different search spaces, it is possible that the execution platform fails to find a suitable service to satisfy the dependencies of an application component.

In Apam, the failure reaction is specified at the architecture level, in the component model. A dependency

 ISBN: 978-1-61208-367-4

declaration can express what should be the policy in case of failure. The currently supported policies are:

- Optional dependency: nothing is done; the field (wire source) will contain "NULL" as target value.
- Wait (duration): the current thread for which a wire could not be resolved is halted until a valid target is found, or until the time limit is reached.
- Exception (name): throws the exception specified by the component.
- Hide: the source component is hidden for all subsequent resolution; all its incoming wires are broken, which may hide its clients and so on.

The hide strategy allows to backtrack the current architecture (as long as dependencies are in the hidden mode) and thus to explore automatically a wide range of possibilities. But since all current architectures must be valid, it is not allowed to remove (hide) components that are explicitly part of the Application architecture.

VI. VALIDATION AND EXPERIENCE

Owing to its flexibility, adaptability and reliability, Apam has been experimented, by academic and industrial teams, as the central layer (often referred as the "dynamic middleware") of two large projects for home automation.

In OpenTheBox project, Apam is mostly used as the central manager for set top boxes, in charge of providing isolation, controlled collaboration between applications[5], including the conflicting accesses to the shared devices [6]. In this case, each application is modeled as a composite, and the contextual properties described in Section IV allow specific policies for each application to be applied.

In the AppsGate project, the set top box is powerful enough to support high level services, advanced functionalities and innovative user interactions. In this project, Apam builds an abstract "model of the world" based on sensors and devices. The high level services are defined as applications at specification and implementation levels, and the execution automatically links the service to the relevant devices.

VII. RELATED WORK

The use of models to represent a system at an appropriate level of abstraction is generalized in software engineering. However, as systems become more dynamic and directly related to the physical world, there is a need to carefully consider the representation relationship, as discussed in [7].

Our approach can be regarded as an example of the general principle of models at runtime [8]: the Apam architectural description is a model of the underlying physical execution. As explained, this model is both an abstract representation (sensitivity) and a prescriptive specification (operationality) of the reality [9].

The abstract Apam application description is a model of the valid space of application's configurations, which evolves by changes at both the execution and component level. Thus, Apam model can be characterized as a "Configuration space and variability model", according to the classification by Vogel et al.[10].

Apam uses architectural models as enabling technology for runtime adaptability. As such, it can be related to many works in dynamic architectures [11][12]. The main idea that we borrowed is that runtime reconfiguration must be reasoned and performed at the architectural level.

If we consider a top-down approach, based exclusively on operationality, the application model is a prescription of the execution, and, the Apam component meta-model can be regarded as an Architecture Description Language. Our meta-model combines the classical concepts of Software Component Models [13] with the intrinsic evolution typical of Service-Oriented Computing [14], in which the concrete architecture is incrementally built as new services are required or made available and bound at execution. In this respect, our proposition can be related to other structural service composition approaches, like SCA [15] or CALM [16], however, these approaches do not define any runtime reconfiguration mechanisms.

We can also think of the Apam runtime platform as a middleware that manages the application execution. Our approach shares then similar goals with reflective middleware platforms [17] that propose an introspection layer that reifies in a causal model the execution elements.

Similarly, some component models propose a reflective runtime to allow introspection and reconfiguration [18][19]. The main difference is that these approaches make the implicit assumption that architecture evolution is an exogenous process, performed by external agents, like administrators or autonomic managers. In our vision, architecture evolution is a continuous, endogenous process, intrinsic to the execution of each application.

Other experimental platforms have been designed specifically for ubiquitous computing. For example, DiaSuite [20] proposes a domain-specific component model to describe the architecture and properties of Sense/Compute/Control applications. The specialized model enables static analysis and verification, beyond what is proposed in Apam, however it doesn't manage runtime dynamicity.

Without surprise, it was the double synchronization (upward and downward) that raised the most difficult technical issues, and the trickiest bugs. Indeed, conflicting changes on the "same" entity can happen "simultaneously" in the model and in the platform. A large fraction of the code is dedicated to solve (reconcile, choose, merge, prevent, notify, etc.) these special cases. It also explains that full causality is difficult to provide, and indeed, is not often provided.

VIII. CONCLUSION

Best practice in software engineering emphasizes the need to work with representations that are simple, homogeneous and at the relevant abstraction level. For that reason, many techniques like levels of abstraction or modeling have been developed. In all case, there is the need to closely control the relationship between the representation and the system represented. Most often, this relationship is operational only: the changes performed on the representation are propagated to the underlying system, supposed to be passive.

 ISBN: 978-1-61208-367-4

The ever increasing interpenetration of the numeric world and our life (smart phones, ubiquitous computing, home automation, etc.) makes abstract representations even more needed, but in the same time the represented system is dynamic, autonomous and its changes are unpredictable. In this case, both the system and its representation are active and both are subject to unpredictable changes, possibly simultaneous and incompatible. Therefore, the representation must be both operational and sensitive, i.e., causal. The realization of a causal representation is very challenging, but at the same time, it is almost needed if reliable applications are to be developed in such a context. The issue we have addressed is a general approach to the development of a causal representation.

In our work, the representation is similar to a traditional model but metaclasses can be explicitly associated with the kind of entities they represent. This association is extensible in the sense that it is implemented in the form of plug-ins: Maven plug-in at development time and Apam causality managers at run time. The platform knows the association and dynamically delegates the causality management to currently plugged-in managers.

In our experimentation, the system represented is an OSGi service platform. It is a limitation because the entities represented are 1) only services, and 2) only those service currently running in OSGi. We have overcome these two limitations by making "everything" a service (proxies, sensors, applications, etc.) and extending the OSGi platform by an extensible provisioning layer, also made of plug-in managers very similar to causal managers. An entity required in the representation layer is automatically deployed in the system (OSGi), and by causality it is created into the representation. The different extensibility mechanisms (causal metaclasses, causal managers, provisioning managers) provide a fairly general framework for the development and management of a causal representation.

The experience shows that causality can be provided systematically and efficiently making much more feasible the reliable development of the new kind of applications like ubiquitous computing.

The Apam platform is available in open source, see [21].

ACKNOWLEDGMENT

Parts of this work have been supported by the European CATRENE project AppsGate and the French ANR "Investissements d'Avenir" project Open-The-Box.

REFERENCES

[1] OSGi Alliance, "OSGi Service Platform Core Specification Release 4", Aug. 2005. [Online]. Available from http://www.osgi.org [retrieved: July, 2014]

[2] E. Damou. "ApAM : A development and execution environment for ubiqutous applications". PhD dissertation. Université de Grenoble, France, Oct. 2013. [In french] [Online]. Available from http://tel.archives-ouvertes.fr/tel-00911462 [retrieved: July, 2014]

[3] C. Escoffier, R. S. Hall, and Ph. Lalanda, "iPOJO: an Extensible Service-Oriented Component Framework", in Proceedings of the International Conference on Services Computing, pp. 474-481, July 2007.

[4] J. Bardin, C. Escoffier, and Ph. Lalanda "Towards an Automatic Integration of Heterogeneous Services and Devices", in Proceedings of the Services Computing Conference, pp. 171-178, Dec. 2010.

[5] J. Estublier and G. Vega. "Managing Multiple Applications in a Service Platform", in Proceeding of the ICSE workshop on Principles of Engineering Service-Oriented Systems, pp. 36-42, June 2012.

[6] J. Estublier, G. Vega, and E. Damou. "Resource Management for Pervasive Systems", in Proceedings of the International Conference on Service Oriented Computing, Lecture Notes in Computer Science, vol. 7759, pp. 368-379, Nov. 2012.

[7] M. Jackson, "Aspects of abstraction in software development", Software & System modeling, vol. 11, no. 4, pp. 495-511, Oct. 2012.

[8] G. Blair, N. Bencomo, and R. B. France, "Models@ run.time", IEEE Computer, vol.42, no.10, pp. 22-27, Oct. 2009.

[9] P-A. Muller, F. Fondement, B. Baudry, and B. Combemale. "Modeling modeling modeling", Software & System modeling, vol. 11, no. 3, pp. 347-359, July 2012.

[10] Th. Vogel, A. Seibel, and H. Giese, "The Role of Models and Megamodels at Runtime", in Proceedings of the Workshop on Models in Software Engineering, Lecture Notes in Computer Science, vol. 6627, pp. 224-238, Oct. 2010.

[11] P. Oreizy and R. Taylor, "On the role of software architectures in runtime system reconfiguration" Software, IEE Proceedings, vol.145, no.5, pp.137-145, Oct. 1998.

[12] J. Magee and J. Kramer, "Dynamic structure in software architectures", in ACM SIGSOFT Software Engineering Notes, vol. 21 Issue 6, pp. 3-14, Nov. 1996.

[13] I. Crnkovic, S. Sentilles, A. Vulgarakis, and M. R. V. Chaudron, "A Classification Framework for Software Component Models", IEEE Transactions on Software Engineering, Vol 37, No. 5, pp. 593-615, Sept. 2011.

[14] J. L. Fiadeiro and A. Lopes, "A Model for Dynamic Reconfiguration in Service-oriented Architectures", in Proceedings of the European Conference on Software Architecture, Lecture Notes in Computer Science, vol. 6285, pp. 70-85, Aug. 2010.

[15] OSOA, "Service Component Architecture Assembly Model V1.00", March 2007. [Online]. Available from http://www.oasis-opencsa.org/sca-assembly [retrieved: July, 2014]

[16] G. Jung and J. Hatcliff, "A type-centric framework for specifying heterogeneous, large-scale, component-oriented, architectures", Science of Computer Programming, vol. 75, no. 7, pp. 615–637, July 2010.

[17] F. Kon, F. Costa, G. Blair, and R. H. Campbell, "The case-for reflective middleware", Communications of the ACM, vol. 45, no. 6, pp. 33-38, June 2002.

[18] M. Léger, T. Ledoux, and T. Coupaye, "Reliable Dynamic Reconfigurations in a Reflective Component Model" in Prooceedings of the International Symposium on Component-Based Sofware Engineering, Lecture Notes in Computer Science, vol. 6092, pp. 74-92, June 2010.

[19] L. Seinturier, Ph. Merle, D. Fournier, N. Dolet, V. Schiavoni, and J-B. Stefani, "Reconfigurable SCA Applications with the FraSCAti Platform" in Prooceedings of the International Conference on Service Computing, pp.268-275, Sept. 2009.

[20] D. Cassou, E. Balland, C. Consel, and J. Lawall. "Leveraging Software Architectures to Guide and Verify the Development of Sense/Compute/Control Applications". in Proceedings of the International Conference on Software Engineering, pp. 431-440, May 2011.

[21] Apam [Online]. http://adeleresearchgroup.github.com/ApAM [retrieved: July, 2014].

 ISBN: 978-1-61208-367-4

Maintaining Vaadin Legacy Applications using DSLs based on Xtext

Marcel Toussaint and Thomas Baar

Hochschule für Technik und Wirtschaft (HTW) Berlin
(University of Applied Sciences)
Treskowallee 8, 10318 Berlin, Germany
Emails: m.toussaint@web.de, thomas.baar@htw-berlin.de

***Abstract*—Vaadin, as a framework for the development of web applications, enables programmers to develop web applications purely in Java. The Vaadin framework has a clean architecture and enjoys a vibrant community. The popularity of Vaadin is certainly also due to numerous tutorials and small examples that illustrate certain aspects of the framework. Vaadin applications, once they became more complex than the appealing tutorials, might run - as well as many other software projects - into maintenance problems. In this paper, we report on a database application, whose programmers followed the suggestions from the tutorials rather strictly. Over time, it became harder and harder to accommodate changes of the database structure since the Java code made certain assumptions on the structure of database tables at many different locations. The classical approach to handle such a situation would be to refactor the entire Java code, which can be very costly. An alternative approach is to use a domain-specific language (DSL) to (a) capture those parts of the application that might vary in future in form of a language, (b) to create a model using this language that reflects the current application, and (c) to change the model due to new requirements and to regenerate those parts of the application that need to be adapted. The last step (c) automates the work of a human software maintainer who would adapt the application code manually due to new requirements (e.g., a new database structure). In this paper, we report on our initial experience when implementing the DSL-based approach using the framework Xtext.**

Keywords–Software design; Metamodelling; Data models; Software maintenance; Graphical user interfaces; Database systems.

I. INTRODUCTION

Web applications became increasingly popular over the last decade due to the multitude of different web browsers, and the convenience for the user of utilizing a web browser in a working environment. One of the main reasons for their popularity is the ability to operate and to update web applications without the need for distributing and installing the software on every single client target platform.

A. *The Open Source Web Application Framework* Vaadin

Vaadin [1] is an open source web application framework for Rich Internet Applications (RIA). Vaadin provides a server side architecture in contrast to JavaScript libraries and browser plugin based solutions. This means, that the majority or even the entire internal program logic is executed on the server. On the client side Vaadin supports Ajax and is based on the framework Google Web Toolkit (GWT) [2]. One of the major advantages of Vaadin is the possibility for the software developer to write the code completely in Java. The framework includes event driven programming and offers Java classes for UI elements such as buttons and lists. In practice, this means writing Vaadin applications is more similar to the development of desktop applications than the traditional Web development with HTML and JavaScript.

Vaadin uses a container-based concept to store and process data objects from external sources (e.g., tables from a database or input files). In this context, a container is a simple entity containing a defined set of items. Each item again possesses a defined set of properties together with their current values.

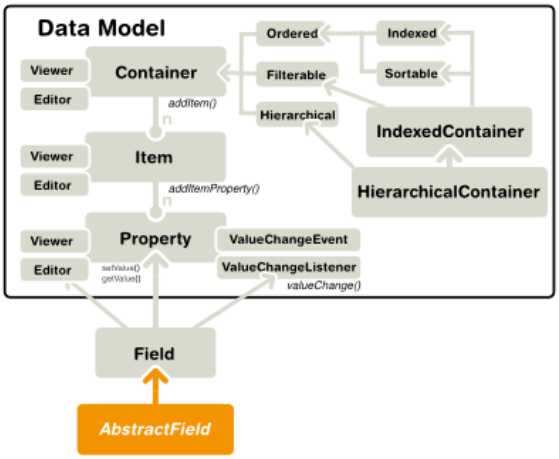


Figure 1. Vaadin's Container Concept [1]

Figure 1 shows Vaadin's architecture to bind UI elements (e.g., subclasses of `AbstractField`) to a `Property` which represents the corresponding attribute value of a data object (e.g., an entry in a database table).

This architecture is very generic and widely applicable, but has the shortcoming that `Property` objects do not have a specific type. The information, which tables exist in the database and what columns they have has to be stored in the application code, together with boilerplate-code to access the attribute values of database entries in a type-safe way. Once the database table structure changes, specific parts of the application code have to be changed as well.

B. *Domain-Specific Languages &* Xtext

Domain-specific languages are specialized computer languages matching a particular problem domain. They permit software design solutions to be expressed using the same

 ISBN: 978-1-61208-367-4

terminology and level of abstraction as the specific problem domain. Over the last few years there has been increased interest in domain-specific languages due to their potential of improving the productivity, quality and especially efficiency of software engineering.

The open-source Xtext framework [3] provides a solid toolkit for developing textual domain-specific languages. To specify a language, a grammar written in Xtext's grammar language has to be created. Once this is done, the user can create models in the specified language. Xtext supports this concept even by generating grammar specific editors. In many cases, the user wishes to generate other artifacts (e.g., source code) from these models. Xtext provides special support for implementing generators to produce such artifacts.

One of the key features of the Xtext framework is the possibility of seamlessly integrating it into the Eclipse IDE in form of a plugin. This plugin provides syntax highlighting for the DSLs created combined with code-folding and -checking. This toolset becomes handy in our scenario, because the Vaadin framework is supported by a corresponding plugin in the Eclipse IDE as well.

C. Tackling Software Maintenance with DSLs

Vaadin's architecture for bridging the gap between UI- and persistence-layer with the very generic classes `Container`, `Item` and `Property` causes the problem, that the programmer has to keep certain information on the database structure elsewhere in the Java code. Ideally this this information is kept in a single place. In practice, however, this is often not the case and the Java code suffers from the smell of *Solution Sprawl* [4].

There are refactoring techniques to eliminate the smell from the code, but refactoring is generally costly [4][5]. Furthermore, it is often not evident that the refactored versions are really better with respect to *performance*, *readability*, *maintenance* of the code. Note that many different quality criteria can be applied to assess the code and some of them might be complementary, i.e., the implementation code cannot match all these criteria perfectly.

Once we accept that some legacy systems cannot be made 'perfect' by refactoring due to the lack of financial and time resources or the lack of skilled programmers we can look for alternatives to deal with the current situation. One observation is, that maintenance requests, e.g., to change the data structure of the underlying database, requires a multitude of adaptations in the application code and thus is considered to be costly. However, these adaptations, that are traditionally done by a programmer, could be automated, if we succeed in

(a) capturing the possible impact maintenance requests might have on the application code (e.g., changing the database structure also implies changing the UI) and in

(b) generating the necessary code changes automatically.

Note that (a) is done by a suitable domain-specific language and that (b) substitutes the work of a human programmer and thus makes a maintenance operation for the legacy code much less costly.

The rest of the paper is organized as follows: In Section II, we dig into details of the legacy application we started from. Throughout the paper, this application will serve as a running example. In Section III, we present our approach on maintaining legacy applications using Xtext DSLs. In Section V, we outline future steps and summarize lessons learned so far.

II. A Motivating Example

At our university, a Web Application is used for management, revision and versioning of course programs. The application has been developed with the Vaadin framework and the developers strictly followed the recommendations presented in several Vaadin showcase projects.

Figure 2 shows a simplified excerpt from the architecture of this application. The database covers a multitude of entities, which are represented by the application via specific domain classes (e.g., the database table `Users` is represented by the domain class `TableUser`). The corresponding application logic and functionality is encoded in associated service classes. Note that in the given example, only a subset of table columns is represented by attributes of the domain class (i.e., the column `password` is not represented). This is due to the fact that some technical columns are not relevant in the context of the applicable service object. The modalities of how and especially the decision, which data should be presented in the user interface, is encoded in a view class `UserView`.

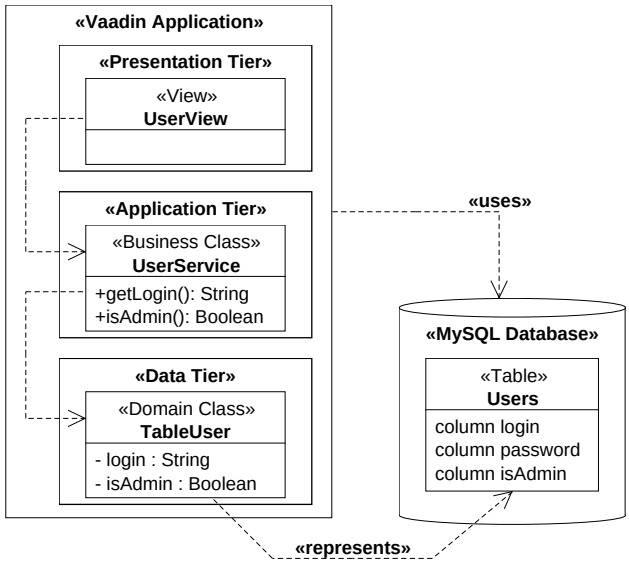


Figure 2. Simplified Excerpt from the Architecture

This implementation leaves us with one major disadvantage concerning the maintainability: Java classes from each tier are rendered deprecated artifacts as soon as the database definition is changed or extended. This produces a large overhead during the development process due to the fact that significant parts of the Java code have to be kept in sync with the database structure.

To illustrate this, consider the following example: if the `Users` table in the database is to be extended by an additional attribute, the Java representation by the domain class `TableUser` has to be updated accordingly to reflect such change. To ensure that the new attribute can be used properly, new functions to manipulate and to retrieve its value(s) must be implemented in `UserService`. Furthermore, the behavior of the corresponding UI class `UserView` concerning the new attribute (i.e., whether the attribute should be displayed or be editable in case of a list object) has to be defined manually.

 ISBN: 978-1-61208-367-4

To counter this problem, the usage of object-relational mapping could provide a proper solution. O/R mapping allows us to automatically convert data from the database content to the Java objects and vice versa. By using object-relational mapping, the concrete domain classes can be adapted automatically, once the database structure has been changed. However, since the UI classes depend on the domain classes, we still would have to manually adapt them to conform to the altered domain classes (and thus to the altered database structure).

III. Using DSLs to maintain Vaadin Applications

We started to explore an alternative approach to solve the maintenance problems described above by using a DSL-based infrastructure. The idea is to generate the domain classes, which represent certain database tables, as well as the corresponding UI classes by using specific code generators. These generators have to regenerate all parts of the application, that need to be adapted due to a maintenance request.

A. Combining multiple DSLs

As maintenance requests often concern several different parts of the application (data, UI), we find it necessary to spread the modeling layer across multiple, interrelated DSLs. Figure 3 illustrates the concept of the 'integration model'.

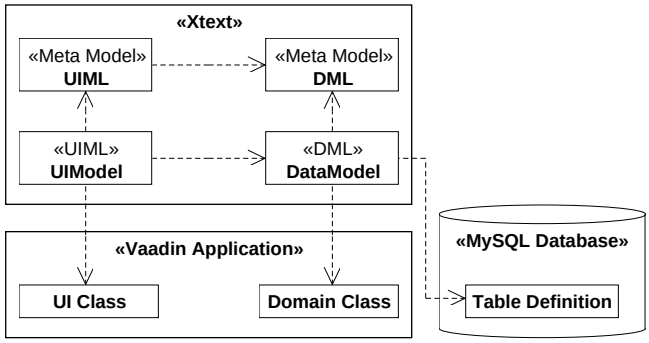


Figure 3. Overview of the Dependencies between Involved Artifacts

With the help of Xtext's grammar language, we created the abstract syntax (i.e., the meta model) for both a UI model language (`UIML`) and a data model language (`DML`) for the existing application. While the data model describes how the data objects are composed and how they are stored, the UI model describes the way these objects are presented to the user. The (`DML`) provides a modeling language for the data objects used in the MySQL context and their representation in the Vaadin application. The concrete code fragments for both the MySQL table definitions and the Java domain classes can be automatically created by using a code generator. Analogously, the Java classes for the UI are regenerated by code generators based on the `UIML`.

B. Designing a DSL for Maintaining Implementation Code

When switching from the traditional approach of maintaining implementation code to our DSL-based approach, one of the biggest challenges is to design an appropriate DSL. This DSL must take into consideration both all information a maintaining request consists of and the current structure of the implementation.

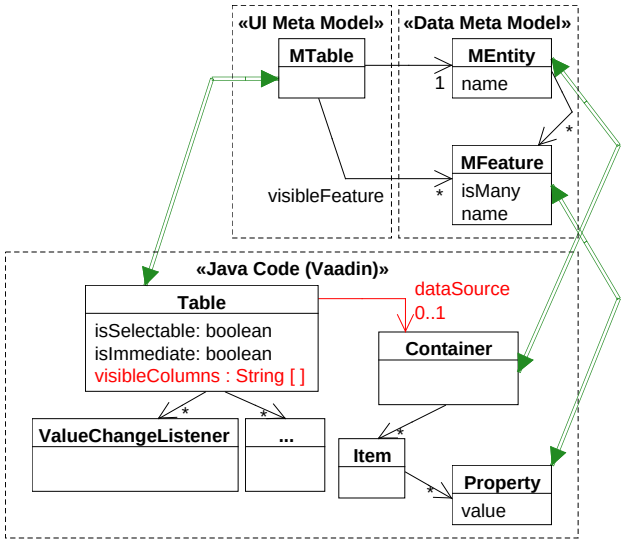


Figure 4. Relationship between DSL and Implementation Code (Excerpt)

The upper part of Figure 4 depictures some content of the meta models for our DSLs `UIML`and `DML`. The lower part presents relevant parts of the application code in form of a UML class diagram. Our meta classes in the meta models always start with a capital 'M'. An instance of `MEntity` consists of many features (`MFeature`) and represents a database table with its columns. The `UIML` consists of those concepts that allow to specify how information is presented in the application UI. For example, the concept `MTable`, which refers to a concrete instance of `MEntity`, represents how entities are displayed in the UI by the Vaadin class `Table`.

The lower part of Figure 4 shows the implementation classes used for displaying database contents. Whenever the content of a database table has to be displayed, an instance of the Vaadin class `Table` is created and configured. The configuration is done in terms of setting attributes such as `isSelectable`, `isImmediate`, `visibleColumns` or in terms of adding configuration objects, e.g., of type `ValueChangeListener` or others. One observation is, that the configurations of `Table` objects remain largely the same, no matter what database entity is to be presented. In our example, only the values for `visibleColumns` and the `dataSource` (marked in red in Figure 4) differ among the instances of `Table`. The `visibleColumns` is a list of strings, containing all columnnames to be shown by the table. The `dataSource` is a `Container` which holds a set of instances of `Item`, which in turn consists of instances of `Property`.

The connection between the upper and lower part of the figure is marked by green arrows. They represent a concrete mapping from model concepts to implementation classes. This mapping encompasses the process of code generation. For example, whenever our code generator processes an `MTable` instance, it produces Java code by which a `Table` instance is created and configured appropriately.

IV. Related Work

The idea of substituting parts of an application written in a conventional programming language such as C, C++ or Java

 ISBN: 978-1-61208-367-4

by one or multiple DSLs in order to increase productivity and to reduce maintenance costs is not new [6]. The goal of *Language-oriented programming (LOP)* [7][8] is to decompose software systems into orthogonal parts that can be described sufficiently detailed by using a DSL. Since the syntax of a DSL is optimized towards a certain purpose, the hope is that maintaining a DSL model is less demanding than maintaining code written in a programming language.

Experience reports on applying LOP for the sake of reducing maintenance costs have been published rarely. Klint et al. report in [9] on a benchmark for the maintenance of different DSL implementations. Some implementation (called vanilla implementations) have been realized using conventional programming languages (Java, JavaScript, C#) while others utilized various DSL tools (ANTLR, OMeta, Microsoft Modeling Platform). The results indicate that the usage of DSL tools is (slightly) advantageous.

Fehrenbach et al. describe in [10] their system *SugarJ* [11] and how a user can embed an external DSL in existing programming code. As an example, they present how the Java Pet Store application can be partly rewritten using four DSLs, what makes the code more readable, type-safe and maintainable. The difference to our work is that we use Xtext instead of SugarJ and that we do not design our DSLs for forward engineering the application from scratch. Instead, our DSLs take the source code from existing legacy applications heavily into account.

V. Conclusion and Future Works

This paper addressed maintaining problems of legacy applications. As an example, we have chosen a Vaadin application, that displays database contents mainly in form of tables. Whenever the structure of the database has changed, the programmer has to adapt the Java classes implementing the application's UI accordingly. Doing this process manually is tedious and error-prone.

We report on the experiences we made, when rewriting the existing web application using DSLs. We focused on those parts of the application, that need to be adapted whenever the structure of the underlying database has changed. We make some suggestions on how a DSL can be designed, such that this DSL exactly covers those parts of the application, that might be affected when adapting the application to new requirements.

In our opinion, the DSL has to be aligned to the existing legacy code. To achieve this, we had to inspect the code and to create corresponding UML class diagrams manually. In future, this might be done automatically by appropriately tailored reengineering techniques.

The main advantage of the proposed solution is that it overcomes potential deficiencies of existing code by replicating and regenerating the code in form of a comprehensible model rather than having to refactor the internal structure of an application. We have chosen the Xtext framework to define DSLs and code generators. This decision has been made due to Xtext's excellent Eclipse integration including syntax highlighting, code completion and static analysis.

References

[1] Vaadin, "Vaadin homepage," retrieved: October 2014. [Online]. Available: http://www.vaadin.com

[2] Google, "Gwt project homepage," retrieved: October 2014. [Online]. Available: http://www.gwtproject.org

[3] Itemis, "Xtext homepage," retrieved: October 2014. [Online]. Available: http://www.eclipse.org/Xtext

[4] J. Kerievsky, Refactoring to Patterns. Addison-Wesley, 2004.

[5] H. M. Sneed, "Planning the reengineering of legacy systems," IEEE Software, vol. 12, no. 1, 1995, pp. 24–34.

[6] P. Hudak, "Modular domain specific languages and tools," in Proceedings of International Conference on Software Reuse (ICSR). IEEE, 1998, pp. 134–142.

[7] M. P. Ward, "Language oriented programming," Software - Concepts and Tools, vol. 15, 1994, pp. 147–161.

[8] M. Fowler, "Language workbenches: The killer-app for domain specific languages?" 2005, retrieved: July 2014. [Online]. Available: http://www.martinfowler.com/articles/languageWorkbench.html

[9] P. Klint, T. van der Storm, and J. Vinju, "On the impact of dsl tools on the maintainability of language implementations," in Proceedings of the Tenth Workshop on Language Descriptions, Tools and Applications, ser. LDTA '10. New York, NY, USA: ACM, 2010, pp. 10:1–10:9. [Online]. Available: http://doi.acm.org/10.1145/1868281.1868291

[10] S. Fehrenbach, S. Erdweg, and K. Ostermann, "Software evolution to domain-specific languages," in Proceedings of Software Language Engineering (SLE), ser. LNCS, M. Erwig, R. F. Paige, and E. V. Wyk, Eds., vol. 8225. Springer, 2013, pp. 96–116.

[11] S. Erdweg, "Sugarj homepage," retrieved: July 2014. [Online]. Available: http://www.sugarj.org

 ISBN: 978-1-61208-367-4

Predicting Change Proneness using Object-Oriented Metrics and Machine Learning Algorithms

Abdullah Al-Senayen, Abdurhman Al-Sahood
Computer Science, College of Computer Science & IT
King Faisal University
Al-Ahsa, Saudi Arabia
{aoaams, bo.dahm.99)@gmail.com

Mohammed Misbhauddin
Information Systems, College of Computer Science & IT
King Faisal University
Al-Ahsa, Saudi Arabia
mmisbhauddin@kfu.edu.sa

***Abstract*— Open Source Software (OSS) has become a huge part of today's software market and a good source for investments. The establishment of the "National Program for Free & Open Source Software Technology" by the top research center (KACST) in Saudi Arabia to encourage the use of OSS within the community is a major motivation to our work. OSS comes with numerous challenges, one of which is constant change. Being able to identify and measure the change proneness in open source software will ensure saving resources like time and effort. In this paper, we measure the capability of classes of machine learning algorithms to predict change proneness in OSS by using object-oriented metrics. Four classes of machine learning algorithms were considered: Probability-based, Function-based, Instance-based and Tree-based. One complete version of the OSS was used as a training set and tested on the subsequent version to predict the change. The machine learning algorithms were compared based on accuracy, specificity, sensitivity and root mean squared error. We found that nearest neighbor algorithm performed better than the other algorithms in terms of sensitivity and specificity. In the future, we plan to test with different parameters to find a better prediction model for software change proneness.**

Keywords-open source software; object-oriented; change proneness; maintainability; prediction.

I. INTRODUCTION

The concept of change is well-known in Software Engineering and the series of changes made to a software system is termed as Software Evolution [1]. The need for evolving software comes because of incorporating new functionality, modifying existing functionality or adapting to new environment conditions etc. However, the impact of this change on the whole system is based on the manner in which the project was developed. According to Güneş Koru and Liu [2], software development can either be closed source or open source. On one hand, closed source projects are well planned and executed. Hence, changes to such systems are localized and can be dealt with in a less haphazard manner. On the other hand, open source projects are developed in an evolutionary manner [3] as a result of which the changes are not restricted and scattered consistently throughout the classes in the project.

OSS has come a long way since the start of its movement in the 1970s. The vision of OSS has changed technology and its market forever. It was the cause of a huge number of breakthroughs. It gave us Google Android, Mozilla Firefox, Linux, Apache, and many more. As OSS changed the world, Saudi Arabia was not an exception. Although the OSS ecosystem in Saudi Arabia is young and developing, it is growing at a fast pace. One aspect of its growth is the huge efforts done by King Abdul-Aziz City for Science and Technology (KACST) actively working to promote the use of OSS in Saudi Arabia. It is running a number of international workshops on the uses of OSS and they are helping in developing standards, awarding innovations, and support academic research on the subject [4].

As OSS development grows in the kingdom and all around the world, we need to consider the characteristics of a good OSS [5]. Extensive research has been conducted over the years to study the relationship between software metrics and various software quality attributes like fault proneness and maintainability [2], [6]-[20]. Around 40-70% of entire cost of a software project is spent on maintenance [17]. The probability that part of software might change is usually referred to as change-proneness. Determining change-prone classes helps in software maintenance, ensuring corrective actions are initiated beforehand. Identifying these classes and the factors that cause these changes is major issue faced during software development. The factors that cause these changes as characterized by Arisholm and Briand [21] as:

1. Structural characteristics of classes (e.g., their coupling).
2. Coding quality of classes.
3. Factors that are captured by the defect history of the classes in the previous release.
4. Change Management team skill and expertise.

In order to identify the causes of change-proneness, we need to identify a rich set of metrics that cover the above-mentioned factors, and hence, help in identifying the exact factors that influence change. In this paper, we are going to look into measures that link the structural characteristics of the classes with their change proneness capability during development. In order to obtain empirical evidence, we analyzed a set of structural metrics and change data that belonged to an open-source project, Heretrix [22]. The change data was extracted comparing classes between consecutive releases of the object-oriented project and the object-oriented metrics from these releases. Metrics were

 ISBN: 978-1-61208-367-4

collected using tools such as ckjm (Chidamber and Kemerer Java Metrics Suite) [23], Dependency Finder [24] and the metric 1.3.6 plug-in in Eclipse [25]. Structural properties of classes, as measured by these metrics, are then associated with change-proneness. In this paper, we are going to treat the aspect of predicting change proneness. Our aim is to measure the capability of certain machine learning algorithms, to predict change proneness in open source software using object-oriented metrics. By being able to predict these classes, we will ensure saving resources like time, money and effort.

The rest of this paper is organized as follows. Section 2 comprises of a detailed literature survey of the various studies done in the past that relate structural properties of classes to their problem source such as change and defect proneness. Section 3 provides the pre-requisite used for the experiment including information on the open source project and the various object-oriented metrics and machine learning algorithms used. The Experimental Setup, Hypotheses and Results are provided in Section 4 and 5, respectively. Results from the experiment are then analyzed with respect to the stated hypotheses in Section 6. Threats that may have affected the validity of the results are highlighted in Section 7. Section 8 concludes the paper and emphasizes on scope for future work based on our findings.

II. RELATED WORK

Gyimothy et al. [6] used open-source software and object-oriented metrics to predict software faults comparing linear and logistic regression models against machine learning algorithms such as decision trees and neural networks. Van koten and Gray [7] used Bayesian Networks as the model to predict maintainability, which is quantified in their approach as the number of lines of changed during a 3-year period. They concluded that by using Bayesian Networks, the model could predict maintainability more accurately than regression-based models. Koru and Liu [2] constructed a tree-based model to predict change-proneness in two large open source projects. They suggested that practitioners should start collecting static metrics and change data to aid their maintenance effort. Zhou and Leung [8] used Multivariate Adaptive Regression Spline (MARS) to predict maintainability.

Eski and Buzluca [9] also used OO metrics to predict change-proneness and its effect on testing effort. Unlike our approach, they used data values from a single version of the software and concluded that change-proneness can be estimated correctly by selecting some optimal set of metrics. Khomh et al. [10] and Romano et al. [11] also proposed a change and fault prediction model but with assessing the impact of anti-patterns rather than OO metrics. Anti-patterns are code patterns with poor design choices.

Lu et al. [12] used statistical meta-analysis techniques to investigate the relationships between OO metrics and change-proneness. Elish and Khiaty [13] also used metrics to predict change-prone classes. In their work, multiple multivariate logistic regression models were built using different sets of dependent and independent variables. They concluded that prediction of change proneness is accurate when product metrics are combined with evolutionary metrics.

Peer and Malhotra [14] used Adaptive Neuro-fuzzy Inference System (ANFIS) to predict change-proneness and compared it against other approaches such as Bagging, Random Forest and Logistic Regression. Malviya and Yadav [15] used k-means clustering and used Chi-Test to decide the cluster with goodness of fit among other clusters.

Research works that compared other machine learning algorithms for their prediction capability like our work recently gained a lot of momentum. Zhu et al. [16] also used OO metrics to predict change-proneness using multiple classification algorithms such as Naive Bayes, C4.5, k -NN, SVM, and an associative classification method. Malhotra and Khanna [17] investigated the effectiveness of logistic regression models against other machine learning algorithms such as Bagging, Random Forest and Multi-layer Perceptron.

Sun et al. [18] go a step forward by assessing a change proposal and the ripple effects caused by it. They used formal concept analysis to assess this effect of change and then proposed a new metric to indicate systems ability to absorb the change. Similarly, Giger et al. [19] went ahead in predicting the type of code change rather than just locating the change-prone parts of a system. While most researchers used software code for change-prediction, Han et al. [20] used design models and defined measures to predict changeability at an earlier stage of software development.

It can be seen from literature that the use of object-oriented metrics to predict change-proneness in open source software is a very active area of research. In this paper, we plan to measure the capability of certain machine learning algorithms, to predict change proneness in open source software using object-oriented metrics. Nevertheless, our research work is different from others in many dimensions:

1. We used a complete version as the training set and then used it over the subsequent version as the testing set to predict the accuracy of the considered algorithms whereas others simply use a single version to build the prediction model [13]-[17].
2. We used classes of machine learning algorithms rather than using a random set of algorithms for comparison.
3. We used baseline prediction models (ZeroR and OneR) to benchmark the evaluation criteria when comparing multiple algorithms.

III. EXPERIMENT SETUP

A. Experiment Subject - Heretrix

We are focusing our research efforts on one particular open-source project, the Heritrix Project [22]. Heritrix is an open-source WebCrawler project started by the Internet Archive in 2003. The software is open source to encourage collaboration and joint development across institutions with similar needs. The Heritrix project almost matches the description of open-source projects: it includes a complete

 ISBN: 978-1-61208-367-4

history of code base, public mailing lists for open discussion, a web site with documentation, and provides release notes for bug tracking. Table 1 provides information regarding the number of releases of the project, the total number of classes and the percentage of classes that changed when compared to its consecutive release. We limited the releases considered in this work until version 2.

TABLE I. CHANGE DISTRIBUTION IN THE RELEASES OF HERITRIX

Version Number	Total No. of Classes	% changed
0.2.0	120	60.83%
0.4.0	164	58.54%
0.6.0	201	35.82%
0.8.0	223	60.54%
0.10.0	246	30.49%
1.0.0	263	45.25%
1.2.0	301	56.81%
1.4.0	369	59.62%
1.6.0	411	17.52%
1.8.0	417	29.50%

B. Object-Oriented Metrics

In this subsection, we present the seventeen metrics that we used to construct the prediction model. Chidamber and Kemerer [26] proposed six of these metrics. We also included some well-known size metrics and number of dependency metrics available from [2]. The definitions of these metrics are shown in Table 2, Table 3, Table 4, Table 5 and Table 6.

TABLE II. SIZE METRICS

Metrics	Description
SLOC	Source lines of code – nonempty and non-comment
NOA	Number of attributes for a class
NOM	Number of methods for a class
NPM	Number of public methods for a class
WMC	Weighted methods per class - sum of the complexities of class's methods.

TABLE III. COHESION METRICS

Metrics	Description
LCOM	Lack of Cohesion in Methods. It counts the sets of methods in a class that are not related through the sharing of some of the class's fields

TABLE IV. INHERITANCE METRICS

Metrics	Description
DIT	Depth of Inheritance – inheritance level from the object hierarchy top
NOC	Number of children – number of immediate descendants of a class

TABLE V. COUPLING METRICS

Metrics	Description
CBO	Coupling between Object Classes – number of classes coupled to a class – can occur through inheritance, function call, return and exceptions.
RFC	Response for a Class - number of different methods that can be executed when an object of that class receives a message
CA	Afferent Coupling - how many other classes use the specific class

TABLE VI. DEPENDENCY METRICS

Metrics	Description
IIP	Inbound Intra-Package Dependencies - number of classes within the same package that depend on this class
IEP	Inbound Extra-Package Dependencies - number of classes in other packages that depend on this class
OIP	Outbound Intra-Package Dependencies Afferent Coupling - number of classes of the same package that this class depends on
OEP	Outbound Extra-Package Dependencies - number of classes of other packages that this class depends on
IIPM	Inbound Intra-Package Method Dependencies - number of methods and fields in other classes of the same package that depend on this class
IEPM	Inbound Extra-Package Method Dependencies - number of methods in other packages that depend on this class

C. Machine Learning Algorithms

In this subsection, we present the six machine learning algorithms that we used to determine the change proneness in the open source software. Of the machine learning algorithms, we used two of them to establish baseline accuracy: ZeroR and OneR algorithms. We selected one machine learning algorithms from four different classes such as Probability-based, Function-based, Instance-based, and Tree-based algorithms.

Baseline Algorithms

- **ZeroR algorithm [28]** is a simple algorithm useful for getting base line performance, in our case accuracy. It ignores all predictors and relies on the target. We used this algorithm to establish baseline accuracy.
- **OneR [28]** creates a rule for each predictor in the data. It then selects the rule with the smallest total error as its one single rule. It constructs a frequency table for each predictor against the target to create a rule for a predictor. We used this algorithm to establish baseline accuracy.

NaiveBayes (Probability-based ML Algorithm)

NaiveBayes algorithm [29] is a probability-based algorithm. It requires only small amount of training set to estimate the variables necessary for explanation. It assumes that the presence or absence of a certain feature is unrelated to the presence or absence of other features. It should be stated that NaiveBayes is based on Bayes theorem.

 ISBN: 978-1-61208-367-4

Multilayer perceptron (Function-based ML Algorithm)

Multilayer perceptron [30] is a function-based algorithm. It maps input data sets into appropriate output data set. It is made up of multiple layer nodes in a directed graph that are fully connected to each other. The network allows signals to travel from the input to the output setting the weights as they propagate through. These weights are tuned for each iteration reducing the overall error for the training set.

Nearest Neighbor (Instance-based ML Algorithm)

Nearest Neighbor (Ibk) [31] is an Instance-based algorithm also known as lazy learning algorithm. It does not do any actual training or learning at first. It populate a sample of the search space with instances whose class is known. When an instance whose class is unknown is presented for evaluation, the algorithm computes its k closest neighbors, and the class is assigned by voting among those neighbors. To prevent ties we use an odd number of k.

J48 (Tree-based ML Algorithm)

J48 is a tree-based algorithm [32] is a class of algorithm that generates a pruned or unpruned C4.5 decision tree and using a divide and conquers strategy to growing the decision tree for each instance. A new unseen instance then traverses the tree until a proper classification is reached.

IV. EXPERIMENT DESIGN

This section describes the design of the experiment. In here, we define the goal of the experiment, the dependent variables and the independent variable and how they were calculated and the tools used in the experiment. This section also gives the procedure of how the experiment was carried out.

A. Goal of the Experiment

- Object of Study: Identify and Characterize change-prone classes
- Purpose: Investigate the correlation between change proneness of a class and the set of structural metrics used in this experiment
- Perspective: From the viewpoint of the researcher and practitioner
- Context: The experiment is conducted with open-source projects and certain measurement tools that are used to calculate the metrics

B. Experimental Variables

The dependent variables in this study are a Boolean variable (Changed) that indicates whether a class changed from one version to another. Any change made to a class during the evolution of a new version from a previous version reflects the change-proneness of that class.

The independent variables are the metrics used to measure the structural properties of the classes. These metrics are presented in section 3-B of this paper.

C. Experiment Hypotheses

Our major objective is to test whether we can predict future changes to a class based on a set of structural metrics. We want to demonstrate that the machine learning algorithms were able to predict change-proneness when compared to baseline algorithms.

We tested the following hypotheses on the case study:

H1: Probability-based algorithms perform better, in terms of Accuracy, Specificity. Sensitivity and Error, than baseline algorithms.

H2: Function-based algorithms perform better, in terms of Accuracy, Specificity. Sensitivity and Error, than baseline algorithms.

H3: Instance-based algorithms perform better, in terms of Accuracy, Specificity. Sensitivity and Error, than baseline algorithms.

H4: Tree-based algorithms perform better, in terms of Accuracy, Specificity. Sensitivity and Error, than baseline algorithms.

We tested the above-mentioned hypotheses by analyzing the relationship between structural metrics of a class from an early version of the system and whether any change occurred to the class during the transition from the early version to a later version.

D. Tools

In order to collect the change data from the system, we used the Beyond Compare 2 [27] tool as a code comparison tool. This tool provided us with information as to whether the code changed from one version to another. Apart from this, we used three measurement tools to obtain the OO metrics. These tools are ckjm [23], Dependency Finder [24] and the metrics 1.3.6 plug-in available for Eclipse IDE [25].

We then used Weka [33] for application of the different machine learning algorithms. Weka is Java-based tool and runs on any platform. The algorithms can either be applied directly to a dataset or called from your own Java code. We applied the stated algorithms in Section 3-C on the Heritrix [22] project metric data collected as the test subject.

E. Experiment Procedure

In our study, we used an OSS to get the classes and run different algorithms through them to get the number of changes in the code compared to different versions of the same class. We uploaded each version of the metrics from Heritrix to Weka and performed different types of algorithms with default settings for each algorithm provided by the tool. We took the number of changes most of the classes were affected with. The steps performed are as follows:

1. **Step1:** All the classes in version n are compared with the corresponding classes in version n+1 to detect changes. This detection is done using a class comparison tool. Based on this information, we populate the Class-change matrix with YES's and NO's depending on whether the class changed from the previous release or not.
2. **Step 2:** All the classes in a version for all the versions are used as an input to a metric calculation tool to calculate all the metrics used as independent

 ISBN: 978-1-61208-367-4

variable in the project. The results from this are then used to populate the Class-metric matrix with appropriate values.

3. S**tep 3:** The input and output is then imported in the data-mining tool Weka for application of the chosen machine learning algorithms. A complete version is used as a training set and the subsequent version as the testing set. This process is repeated for all the versions.
4. **Step 4:** The results of the algorithms in terms of accuracy, root-mean squared errors, sensitivity, and specificity are recorded and compared as shown in the next section.

V. RESULTS

In this work, we collected four result values and used them for comparing the various machine learning algorithms.

A. Accuracy

Accuracy is the percentage of how accurate the algorithm is in predicting the change-proneness of a class based on the OO metrics input. The accuracy of all the selected algorithms across all versions and their average accuracy is shown in Table 7.

TABLE VII. ACCURACY OF THE MACHINE LEARNING APPROACHES

	ACCURACY					
	ZeroR	**OneR**	**Naïve Bayes**	**MLP**	**IBk**	**j48**
v0.2-v0.4	58.5	69.5	80.0	71.3	64.6	71.3
v0.4-v0.6	35.8	60.7	69.7	72.6	67.2	67.2
v0.6-v0.8	39.5	62.3	57.4	59.2	62.3	62.0
v0.8-v0.10	30.5	57.0	74.0	63.0	61.0	57.3
v0.10-v1.0	54.8	63.5	64.3	66.9	70.3	68.4
v1.0-v1.2	43.2	62.1	57.8	58.5	61.8	66.1
v1.2-v1.4	59.6	62.0	53.1	62.0	61.8	62.6
v1.4-v1.6	17.5	52.6	77.6	59.1	55.5	57.9
v1.6-v1.8	70.5	74.0	75.5	74.6	76.0	75.3
	45.5	62.6	67.7	65.2	64.5	65.3

a. Values are percentages

B. Specificity

Specificity is the percentage of the values that were originally “No” and also predicted as “No” as obtained from the confusion matrix. After applying the algorithms, the result was that the baseline specificity has a fair specificity better than the other algorithms specificity. ZeroR algorithm was the best giving perfect specificity for five times, as shown in Table 8.

TABLE VIII. SPECIFICITY OF THE MACHINE LEARNING APPROACHES

	SPECIFICITY					
	ZeroR	**OneR**	**Naïve Bayes**	**MLP**	**IBk**	**j48**
v0.2-v0.4	100	54.4	33.8	44.1	39.7	33.8
v0.4-v0.6	100	44.2	18.6	24.8	43.4	39.5
v0.6-v0.8	0	13.6	4.5	14.8	9.1	14.8
v0.8-v0.10	100	56.1	17.5	44.4	45.6	50.9
v0.10-v1.0	0	11.8	7.6	7.6	11.8	9.7
v1.0-v1.2	0	16.9	10.0	10.8	24.6	20.0
v1.2-v1.4	100	47.0	13.4	37.6	36.2	38.9
v1.4-v1.6	100	54.9	15.6	43.4	49.6	45.7
v1.6-v1.8	0	2.0	5.1	1.7	7.5	3.1
	55.6	33.4	14.0	25.5	29.7	28.5

a. Values are percentages

C. Sensitivity

Sensitivity is the percentage of values, which were “yes” and also predicted as “yes” as obtained from the confusion matrix. After applying the algorithms, the result was that the baseline sensitivity has a fair sensitivity better than the other algorithms sensitivity. ZeroR algorithm was the best giving perfect sensitivity for five times. Table 9 shows the sensitivity result of all the algorithms on the given versions.

TABLE IX. SENSITIVITY OF THE MACHINE LEARNING APPROACHES

	SENSITIVITY					
	ZeroR	**OneR**	**Naïve Bayes**	**MLP**	**IBk**	**j48**
v0.2-v0.4	100	86.5	89.6	82.3	67.7	75.0
v0.4-v0.6	100	69.4	48.6	68.1	86.1	79.2
v0.6-v0.8	0	46.7	32.6	42.2	43.7	46.7
v0.8-v0.10	100	86.7	54.7	80.0	76.0	76.0
v0.10-v1.0	0	33.6	30.3	36.1	48.7	42.0
v1.0-v1.2	0	46.2	33.3	35.1	51.5	55.6
v1.2-v1.4	100	68.2	30.5	61.8	60.5	63.6
v1.4-v1.6	100	87.5	45.8	70.8	79.2	75.0
v1.6-v1.8	0	16.3	29.3	17.9	36.6	23.6
	55.6	60.1	43.8	54.93	61.1	59.6

a. Values are percentages

D. Root Mean Squared Error (RMSE)

RMSE is the difference between values predicted by a model and the values actually observed. After applying the algorithms, the result was that the Multilayer Perceptron

 ISBN: 978-1-61208-367-4

algorithm had the lowest RMSE. Table 10 shows the RMSE result of all the algorithms on the given versions.

TABLE X. RMSE OF THE MACHINE LEARNING APPROACHES

	ROOT MEAN SQUARED ERROR					
	ZeroR	**OneR**	**Naïve Bayes**	**MLP**	**IBk**	**j48**
v0.2-v0.4	0.49	0.55	0.45	0.47	0.59	0.51
v0.4-v0.6	0.53	0.63	0.54	0.48	0.57	0.55
v0.6-v0.8	0.55	0.61	0.65	0.59	0.61	0.59
v0.8-v0.10	0.55	0.66	0.50	0.56	0.62	0.61
v0.10-v1.0	0.52	0.60	0.58	0.48	0.54	0.54
v1.0-v1.2	0.51	0.62	0.64	0.51	0.61	0.53
v1.2-v1.4	0.49	0.62	0.68	0.48	0.62	0.53
v1.4-v1.6	0.57	0.69	0.46	0.55	0.66	0.61
v1.6-v1.8	0.47	0.51	0.49	0.44	0.49	0.46
	0.52	0.61	0.55	0.51	0.59	0.55

VI. DISCUSSION

In this section, we will discuss our four comparison criteria used. First, we have accuracy. After applying the algorithms, the result was that all of the machines learning algorithms have a fair accuracy better than the baseline accuracy. Naive Bayes and Nearest Neighbor are better considering both have given the comparatively better results than the most and Naïve Bayes gave the highest percentage with 68% as shown in Table 7. Then, there is specificity. After applying the algorithms, the result was that the baseline specificity has a fair specificity better than the other algorithm's specificity. ZeroR algorithm was the best giving perfect specificity for five times as shown in Table 8. Next is Sensitivity. After applying the algorithms, the result was that the nearest neighbor algorithm was the only one with a better average sensitivity percentage compared to the baseline algorithm as shown in Table 9. Finally, we have RMSE. After applying the algorithms, the result was that the Multilayer Perceptron algorithm had the lowest RMSE, which was better than the baseline algorithm as shown in Table 10. Based on this analysis, we reject all the hypotheses H1, H2, H3 and H4 as no class of machine learning algorithm performs better than the baseline algorithms in terms of all considered comparison factors: accuracy, sensitivity, specificity and error.

VII. THREATS TO VALIDITY

This section discusses the threats to validity in this study and the way they were treated throughout the experiment.

A. Construct Validity

Construct Validity is the degree to which the independent variables and dependent variables accurately measure the concepts they purport to measure. The dependent variables we used in our study were change, which is a Boolean variable as to whether the class changed or not. As the way they are computed is straightforward, we consider them constructively valid. If any, the way the size was calculated can pose a slight threat if there is a better way for it.

B. External Validity

External Validity is the degree to which the results of the research can be generalized to the population under study and other research settings. A crucial threat lies with the size of the case study considered. Only a single project releases are considered in this study with 10 releases. This may affect the generalization of the identified conclusions. On the same lines, another valid threat that cannot be excluded until extensive empirical results are collected is that the case study will reflect the characteristics from a specific domain. In addition, the data collected from the open source project was by analyzing the code. Poor documentation can affect the results of the analysis significantly.

C. Internal Validity

Internal Validity is the degree to which conclusions can be drawn about the casual effect of independent variables on the dependent variables. Apart from the variables considered, our approach might have omitted other important variables that can serve as predictors. In addition, the size of the open source project can be considered as a potential threat as our project was not very big, but significantly large.

VIII. CONCLUSION AND FUTUREWORK

This paper reported findings of an empirical study conducted to investigate the measures that affect the change-proneness of classes in an open source project. The goal was to use a case study from the open source community in order to explore the relationship between the structural characteristics of the project and the change proneness of classes within that project from one version to the other. The study concluded a lot of interesting results that conform to previous studies, such as size-related metrics and coupling metrics are correlated with change proneness. In conclusion to the experiment, we believe that using machine learning algorithms to predict change proneness in open source software using object-oriented metrics is an excellent field for research and needs to be further investigated. In many cases, we were able to identify that the baseline accuracy performed better than the machine learning algorithms considered. This result calls for more research for better algorithms that can be used for prediction of change-proneness.

It should be noted that we used the default setting of Weka for all the machine-learning algorithms used. Moreover, as a future step, we plan to change the settings of certain parameters in these algorithms to find a better prediction model for software change proneness. In addition, we plan to make use of correlation and principal component analysis to select only those metrics that seem to affect the change-proneness. In addition, the study provided some useful information regarding dependency relationships and their association. Based on our findings, we suggest

 ISBN: 978-1-61208-367-4

practitioners dealing with open source projects to collect static metrics and change data as part of their development effort. This data can be used to prioritize preventive action on the classes that are still under development.

ACKNOWLEDGMENT

The authors acknowledge the support of King Faisal University in the development of this work.

REFERENCES

[1] M. M. Lehman and L. A. Belady, Program evolution: processes of software change. CA: Academic Press Professional, Inc., 1985.

[2] A. Güneş Koru and H. Liu, "Identifying and characterizing change-prone classes in two large-scale open-source products," Journal of Systems and Software, vol. 80, Jan. 2007, pp. 63-73, doi: 10.1016/j.jss.2006.05.017.

[3] E. S. Raymond, The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. CA: O'Reilly Media, Inc., 2001.

[4] King Abdulaziz City for Science and Technology, National Program for Free & Open Source Software Technology. Available: http://www.motah.org.sa. Retrieved: August, 2014.

[5] C. Årdal, A. Alstadsæter, and J.-A. Røttingen, "Common characteristics of open source software development and applicability for drug discovery: a systematic review," Health Research Policy and Systems, vol. 9, Sept. 2011, pp. 1-16, doi:10.1186/1478-4505-9-36.

[6] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction," IEEE Trans. Softw. Eng., vol. 31, Oct. 2005, pp. 897-910, doi: 10.1109/tse.2005.112.

[7] C. van Koten and A. R. Gray, "An application of Bayesian network for predicting object-oriented software maintainability," Inf. Softw. Technol., vol. 48, Jan. 2006, pp. 59-67, doi: 10.1016/j.infsof.2005.03.002.

[8] Y. Zhou and H. Leung, "Predicting object-oriented software maintainability using multivariate adaptive regression splines," Journal of Systems and Software, vol. 80, Aug. 2007, pp. 1349-1361, doi: 10.1016/j.jss.2006.10.049.

[9] S. Eski and F. Buzluca, "An Empirical Study on Object-Oriented Metrics and Software Evolution in Order to Reduce Testing Costs by Predicting Change-Prone Classes," Proc. IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW 11), IEEE, Mar 2011, pp. 566 – 571, doi: 10.1109/icstw.2011.43.

[10] F. Khomh, M. Penta, Y.-G. Guéhéneuc, and G. Antoniol, "An exploratory study of the impact of antipatterns on class change- and fault-proneness," Empirical Software Engineering, vol. 17, June 2012, pp. 243-275,doi: 10.1007/s10664-011-9171-y.

[11] D. Romano, P. Raila, M. Pinzger, and F. Khomh, "Analyzing the Impact of Antipatterns on Change-Proneness Using Fine-Grained Source Code Changes," Proc.19th Working Conference on Reverse Engineering (WCRE 12), IEEE Computer Society,Oct. 2012, pp. 437-446, doi: 10.1109/WCRE.2012.53.

[12] H. Lu, Y. Zhou, B. Xu, H. Leung, and L. Chen, "The ability of object-oriented metrics to predict change-proneness: a meta-analysis," Empirical Software Engineering, vol. 17, June 2012, pp. 200-242,doi: 10.1007/s10664-011-9170-z.

[13] M. O. Elish and M. Al-Rahman Al-Khiaty, "A suite of metrics for quantifying historical changes to predict future change-prone classes in object-oriented software," J. Softw. Evol. and Proc., vol. 25, May 2013, pp. 407-437,doi: 0.1002/smr.1549.

[14] A. Peer and R. Malhotra, "Application of adaptive neuro-fuzzy inference system for predicting software change proneness," Proc. Advances in Computing, Communications and Informatics (ICACCI 13), IEEE, Aug. 2013, pp. 2026-2031, doi: 10.1109/ICACCI.2013.6637493.

[15] A. K. Malviya and V. K. Yadav, "Maintenance activities in object oriented software systems using K-means clustering technique: A review," Proc. CSI Sixth International Conference on Software Engineering (CONSEG 12), IEEE, Sept. 2012, pp. 1-5, doi: 10.1109/CONSEG.2012.6349490.

[16] X. Zhu, Q. Song, and Z. Sun, "Automated Identification of Change-Prone Classes in Open Source Software Projects," Journal of Software, vol. 8, Feb. 2013, pp. 361-366, doi: 10.4304/jsw.8.2.361-366.

[17] R. Malhotra and M. Khanna, "Investigation of relationship between object-oriented metrics and change proneness," International Journal of Machine Learning and Cybernetics, vol. 4, Aug. 2013, pp. 273-286, doi: 10.1007/s13042-012-0095-7.

[18] X. Sun, B. Li, and Q. Zhang, "A Change Proposal Driven Approach for Changeability Assessment Using FCA-Based Impact Analysis," Proc. IEEE 36th Annual Computer Software and Applications Conference (COMPSAC 12), IEEE, Jul. 2012, pp. 328 – 333, doi: 10.1109/COMPSAC.2012.44.

[19] E. Giger, M. Pinzger, and H. C. Gall, "Can we predict types of code changes? An empirical analysis," Proc. 9th IEEE Working Conference on Mining Software Repositories (MSR 12), IEEE, Jun. 2012, pp. 217-226, doi: 10.1109/MSR.2012.6224284.

[20] A.-R. Han, S.-U. Jeon, D.-H. Bae, and J.-E. Hong, "Measuring behavioral dependency for improving change-proneness prediction in UML-based design models," Journal of Systems and Software, vol. 83, Feb. 2010, pp. 222-234, doi: 10.1016/j.jss.2009.09.038.

[21] E. Arisholm and L. C. Briand, "Predicting fault-prone components in a java legacy system," Proc. ACM/IEEE International Symposium on Empirical Software Engineering (ISESE 06), ACM, 2006, pp. 8-17, doi: 10.1145/1159733.1159738.

[22] P. Jack. Heritrix. Available: https://webarchive.jira.com/wiki/display/Heritrix/Heritrix, Retrieved: Sept. 2014.

[23] D. D. Spinellis. (2008). ckjm — Chidamber and Kemerer Java Metrics. Available: http://www.spinellis.gr/sw/ckjm/, Retrieved: Sept. 2014.

[24] J. Tessier. Dependency Finder. Available: http://depfind.sourceforge.net/, Retrieved: Sept. 2014.

[25] F. Sauer, Metrics 1.3.6. Available: http://metrics.sourceforge.net/, Retrieved: Sept. 2014.

[26] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," IEEE Trans. Softw. Eng., vol. 20, Jun. 1994, pp. 476-493, doi: 10.1109/32.295895.

[27] Scooter Software. Beyond Compare 2. Available: http://www.scootersoftware.com/, Retrieved: Sept. 2014.

[28] I. H. Witten, E. Frank, M. A. Hall, Data Mining: Practical Machine Learning Tools and Techniques, MA: Morgan Kauffman Publishers, 2011.

[29] D. Heckerman, D. Geiger, D. M. Chickering, "Learning Bayesian networks: The combination of knowledge and statistical data," Machine Learning, vol. 20(3), Sept. 1995, pp. 197–243, doi: 10.1007/BF00994016.

[30] C. M. Bishop, Neural networks for pattern recognition, New York: Oxford University Press, 1995.

[31] D. Aha, "Tolerating noisy, irrelevant, and novel attributes in instance-based learning algorithms," International Journal of Man-Machine Studies, vol. 36(2), Feb. 1992, pp. 267–287, doi: 10.1016/0020-7373(92)90018-G.

[32] J. Quinlan, C4.5: Programs for Machine Learning. CA: Morgan Kaufmann, 1993.

[33] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software: An Update," SIGKDD Explorations, vol. 11, Jun. 2009, pp. 10-18, doi: 0.1145/1656274.1656278.

 ISBN: 978-1-61208-367-4

Towards an Efficient Traceability in Agile Software Product Lines

Zineb Mcharfi, Bouchra El Asri, Ikram Dehmouch
IMS Team, SIME Laboratory
ENSIAS, Mohammed V Rabat University
Rabat, Morocco
{zineb.mcharfi@gmail.com, elasri@ensias.ma, ikram.dehmouch@gmail.com}

***Abstract*—In a volatile market, where it is difficult to predict future needs, classical Software Product Lines show limitations and become pricey. Therefore, researchers managed to add supplements in order to reach flexibility, and this led to the Agile Product Line Engineering concept. However, this concept has not gained yet sufficient maturity, and works are still necessary to establish the best practices for putting Agile Product Line Engineering into practice, especially when it comes to their traceability. In this paper, we discuss the correlation between agility and traceability dimensions through the state of the art of traceability in Agile Software Product Lines, and present our solution based on markers and break-even point in order to establish a traceability methodology in Agile Software Product Lines.**

Keywords-Software Product Lines; Agile Software Product Lines; traceability; efficient traceability.

I. INTRODUCTION

Considering market growth and competitiveness, companies try to achieve mass customization with lower costs, reduce time to market, and insure product quality while getting customer's satisfaction. From a software engineering point of view, Software Product Lines (SPL) is a promising concept that helps dealing with those challenges [1][2].

However, in some business environments, SPL may not be enough reactive compared to market growth. In fact, designing a SPL requires deploying important efforts and time in order to speculate on future products and functionalities that may be needed. Also, the Return On Investment (ROI) of those efforts might be very small in a volatile market [3]. Those constraints pushed developers and researchers to look for improving SPL in order to gain flexibility, which led to the concept of Agile Product Line Engineering (APLE) [4]–[6].

Many researchers worked on the feasibility of combining SPL and Agile Software Development (ASD) [3]–[6], as both of them share the same objectives of increasing productivity and software quality while optimizing production time, even if they present differences in the concept and practices [4]. Traceability might be considered as one of the challenging points in combining SPL and agility; the former, because of its complexity and need to manage variability, requires traceability documentation to assure consistency of the links between artifacts and facilitate changes implementation [2], while the latter advocates less use of documents [7].

In the present paper, we will illustrate, throughout a state of the art, how the existing works manage traceability in their Agile Software Product Lines (ASPL), depending on the agile method used. We will also present our contribution, a methodology based on the concepts of "markers" and "break-even point" for an efficient traceability in ASPL.

The remainder of this paper is structured as follow: in Sections II, we describe the concepts of SPL, ASD and ASPL. Section III presents the traceability in SPL and a state of the art of traceability in ASPL. We discuss our contribution in Section IV and illustrate it in a case study in Section V, before concluding in Section VI.

II. BACKGROUND AND MOTIVATIONS

In this section, we will first briefly introduce SPL and ASD in order to present later the ASPL, a concept based on the combination of the two previous ones.

A. Software Product Lines

As defined by Northrop [1], a SPL is *"a set of software-intensive systems that share a common, managed feature set satisfying a particular market segment's specific needs or mission and that are developed from a common set of core assets in a prescribed way"*. It is used in the organizations that produce numerous products answering specific needs, but having many components in common. Those common components (e.g., architecture, requirements, test plans, schedules, budgets and processes description) are called "core assets". Adopting a SPL approach allows to produce new systems by reusing the existing ones, in an organized manner.

Accordingly, SPL is a combination of three major interacting elements, called the SPL essential activities [1][8]: (1) core asset development or Domain Engineering (DE), (2) product development or Activities Engineering (AE) and (3) technical and organizational management that orchestrates those two activities.

 ISBN: 978-1-61208-367-4

SPL is by far considered as an up-front, proactive (in opposite to reactive) reuse demarche [9]: it is based on a production plan, involves both technical and organizational management, is a direct consequence of the organization strategy, and it is used to reach predictable results.

B. Agile Software Development

ASD is a concept based on the Agile Manifesto [7]. As for SPL, ASD seeks to satisfy customer needs rapidly, while insuring a good software quality, yet unlike SPL, the ASD concept is based on simplicity, iterations, and reducing up-front design [5].

ASD values, described in the Agile Manifesto, are "individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation and responding to change over following a plan" [7].

The Agile Manifesto defines also twelve principles for ASD [7]. Hereinafter some: (1) customer satisfaction by rapid delivery of useful software, (2) welcome changing requirements, even late in development and (3) regular adaptation to changing circumstances.

Thus, ASD shows values where SPL shows weaknesses, especially when it comes to flexibility and adaptation to changing requirements and circumstances.

Accordingly, some complementarity can be found between SPL and ASD, which led to the APLE concept.

C. Agile Product Line Engineering: Software Product Lines combined to Agile Software Development

As explained earlier, SPL need an up-front design, with heavy processes and significant efforts. It helps answering planned changes, but if it comes to unstable environments with rapidly changing conditions, the investment in SPL might be pricey [3]. On the other hand, ASD seeks to satisfy customer requirements in a reactive way, promoting continuous discussion with the customer, and avoiding up-front developments.

According to Díaz et al. [3] and Ghaman et al. [5], the combination of SPL and ASD principles allows eliminating long term investment in up-front design, especially in volatile markets where it would represent a non-profitable investment in the long term with huge losses due to no-longer useful core assets or never used ones. It allows also dealing with situations where there is lack of knowledge about domain engineering, or where no speculation can be made.

Many works discuss the application of agility to SPL: In [6], agility is used in the design phase and the benefices of its introduction by gaining in speed are demonstrated. Noor et al. [10] used a collaborative approach to introduce agility when planning and scooping the Product Line (PL). They used some agile development principles, such as valuing customer collaboration and high degree of flexibility. Urli et al. [11] described the application of agility for SPL evolution through a case study, using Composing Feature Models (CFM); they first built an information broadcasting system for a limited academic structure, but then had to deal with larger institutions and numerous customers, which represented multiple devices and sources of information. Therefore, they used a SPL demarche for the re-engineering of their system and, as they had to interact continuously with the customers, they sought lightness and introduced agility to their approach. This decision helped them reach simplicity (they decomposed the requirements in features with fine granularity) and be more reactive to the customers' needs. Another approach was established by Ghanam and Maurer [12], who used a Test Driven Development (TDD) method to deal with agility in SPL. They introduced SPL demarche in an agile environment that uses eXtreme Programing (XP), and instead of using requirement documents to begin development, they used Acceptance Tests (AT) generated through the XP process as test artifacts, which are the basis for the model adopted.

III. Traceability in Agile Software Product Lines

In such a complex environment (i.e., ASPL), where we have to manage variability in a constantly evolving context, it is very important to insure traceability along the software development process.

However, based on the observation made by the review in [3], and completed with our literature analysis, we noticed that very few researches deal explicitly with the problematic of traceability in ASPL, knowing that managing traceability is very important in such evolving environments. Therefore, we choose to discuss the problematic of traceability in ASPL, given the challenges that it presents.

A. Traceability in Software Product Lines

Traceability helps follow the components' life, link between different software artifacts, from requirements to source codes and backwards and, in a larger scale, helps verify that all requirements have been implemented and the artifacts documented [13]. It is also a mean to consider different architecture choices and identify errors, and to facilitate communication between stakeholders [14]. Traceability is very helpful when it comes to maintenance and evolution as it allows analyzing and controlling the impact of changes [15].

SPL add complexity to the traceability due to their reuse characteristics and the variability management [16]. Berg et al. [17] proposes a conceptual variability model to deal with traceability in SPL and consider that, in addition to the two dimensions of traceability in a simple software (i.e., phases of development and levels of abstraction), for SPL there is need to add variability as a third dimension. They propose to handle SPL variability, and especially the traceability problematic, by adopting a three dimensions conceptual variability model that uses feature modeling to manage variability and traceability. Anquetil et al. [14][16] added a fourth dimension, namely evolution, to link between the different versions of every artifact, and a fifth one, versioning, to trace components' changes in time.

In the next section, we will draw up a state of the art of traceability in ASPL, based on the five traceability dimensions, as presented by Anquetil et al. [14]: (1) refinement traceability that links abstract artifacts to more concrete ones that realize them (no variability), (2) similarity

 ISBN: 978-1-61208-367-4

traceability for links between artifacts at the same level of abstraction (requirements, design, etc.), (3) use-variability traceability for instantiation links (from DE to AE), (4) realize-variability traceability to link between the variant and the artifact that realizes it at the DE level, (5) versioning traceability to link two successive versions of an artifact.

B. State of the art of traceability in Agile Software Product Lines

In this section, we will draw up a listing of works that present a methodology for introducing agility in SPL, and discuss those methods according to the following questions: (1) What are the traceability dimensions (according to [14]) does the presented methodology cover? (2) Which agile method is used? (3) At which stage of SPL is the agility introduced? (4) How does it deal with traceability?

Our first observation is that not all the methodologies found in literature propose solutions that take into consideration traceability (Table I). One assumption might be that it depends on the stage where agility is used. In fact, in [24], agility was applied in scoping, and all the work was focalized on it. In the other papers, at least refinement traceability is covered.

Papers that approach the architectural problematic and variation points [12][18][22][23] cover another dimension: variability traceability, with a link type "realize". The approach presented in [12][18], which uses acceptant tests, and the one that combines workflow and web services [22] handle also the variability traceability with a link type "use".

TABLE I. WORKS ANALYSIS ACCORDING TO TRACEABILITY DIMENSIONS

Traceability dimensions				
Refinement	***Similarity***	***Variability (Use)***	***Variability (Realize)***	***Versioning***
[10][12][18]–[22]	-	[12][18][22]	[12][18][22][23]	[19]

TABLE II. WORKS ANALYSIS ACCORDING TO THE AGILE METHOD

Reference	***Agile method***	***Level of agility application***	***How traceability is applied***
[23]	Scrum	Architecture	Using Product Line Architectural Knowledge (PLAK) metamodel and Design decision by documenting adding features and changing features
[12][18]	XP	Requirements	AT
[24]	Agility principles	Scoping	-
[10]	Agility principles applied though Collaboration Engineering	planning	ThinkLets + collaborative process
[19]	Evo	Requirements management	Impact Estimation Tables (IET)
[20][21]	Agility principles and XP at « Preparing for Derivation » phase	Product Derivation	-
[26]	Some agile principles (Flexible, quick, adaptable, user-oriented)	Design to architecture	WebServices + workflow + WebPads-based approach

In [19], versioning traceability is addressed through the use of Impact Estimation Tables. Iterations (and accordingly components changes) are listed for each goal per project and per release.

In general, there is a lack in covering several traceability dimensions in ASPL approaches literature. Also, concerning the agile methods (Table II), many works are based only on agility principles [10][20][24]. XP approach is also widely used [12][18][20][21]. However, by using AT, [12] and [18] cover three of the five traceability dimensions and propose an approach that covers the entire process, from requirements to code units.

IV. OUTLOOK AND CONTRIBUTION

Based on our researches, we found the study of ASPL a challenging field that did not gain yet sufficient maturity, especially when it comes to managing traceability. In fact, in case of ASPL, we need to consider the agile characteristics of the environment. Adding agility means frequent requirements' change, even late in development, and continuous interaction with customers. Also, while agility tries to avoid heavy processes and excessive documentation, traceability needs more produced and maintained documents.

In the works related to ASPL, as discussed in the previous section, there is lack of managing traceability: not all the ASPL methodologies proposed in literature deal with traceability and, for those taking it into consideration, the agile configuration proposed doesn't allow tracing the whole PL chain, according to the five traceability dimensions detailed in [14].

Moreover, referring to our literature analysis, we noted that only papers presenting an automated refactoring approach used traceability in an efficient way: only really affected elements in the SPL are localized and modified before rebuilding the SPL [12][18][19].

Thus, for our contribution, we propose a methodology based on markers for efficient traceability in an ASPL environment: in a SPL, every produced element is the result of specific concatenation and instantiation of some product line components. Knowing this combination helps tracing efficiently the product generation path by targeting only the concerned components. Based on this observation, we are establishing an approach that consists of adding a marker to every SPL component. Each marker is unique and encapsulates the component characteristics and, as a product is the result of specific core assets instantiation, it inherits from those core assets' characteristics. Therefore, the idea is to identify the product with a marker composed from the corresponding core assets' ones, and to create a link between the components and the products, based on those markers. The marking step is added to the core assets generation

 ISBN: 978-1-61208-367-4

process (see Figure 1). However, a special treatment is reserved to variation points. In fact, variants share characteristics with their parent components (i.e., variation points). Thus, instead of generating a new marker for the variants, we use a function for "marker mutation". This function allows modifying the variation point marker to generate the variant's one, while keeping the former's characteristics and adding the latter's specific ones. It helps lighten the process, as we are in an agile environment, and establishing a realize-variability traceability link. By the end of this process, a multidimensional marking matrix is generated. Its dimensions correspond to core assets and its cells to a combination of the corresponding markers. Thus, each derived product is distinguished by a marker that corresponds to a specific cell in the multidimensional matrix (see Figure 2). However, in order not to complicate the matrix and to preserve the agility of the environment, (tracing the whole product generation process might be heavy and costly, and even useless regarding the traceability purpose in the developed ASPL), we introduce the concept of "break-even point". It represents the point of balance between the desired level of traceability detail and the costs of building and maintaining the system. It is flexible and depends on the level of traceability needed. The aim of this break-even point concept is to define a traceability limit based on which we select only the core assets needed for the product traceability. The product marker is then assembled depending on the composition of those core assets in the product. In order to define the parameters to consider for establishing a break-even point, we are conducting a study to outline the limitations of traceability in an agile environment. We aim to determine, through this study, the level of traceability that does not penalize the agility of the ASPL, and we intend to evaluate this approach using graph theory principles, to prove that the selected subgraph (connection of core assets) can effectively allow tracing the products' generation paths, knowing the environment constraints.

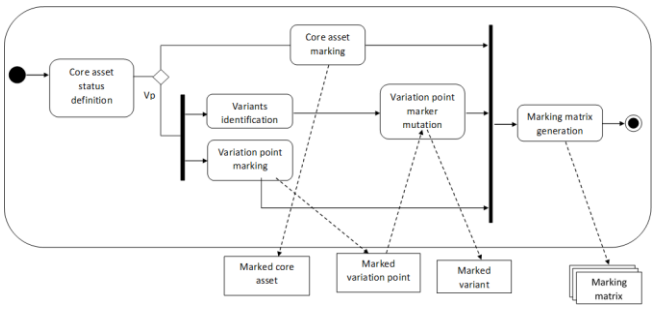


Figure 1. Core assets marking process

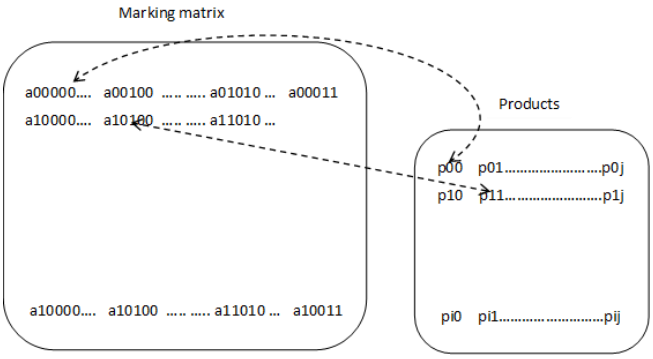


Figure 2. Link between products and marking matrix

V. CASE STUDY

To illustrate our approach, we present hereafter a case study of offers implementation in the case of a telecommunication operator.

Telecommunication market is very competitive and each operator has to be reactive to the market changes. Also, with the expansion of smartphones and intelligent home equipment, trend is for broadband, high speed data transmission, and free short messages and calls. Therefore, offers share the same objectives but present them in different ways, depending on the proposed services, the pricing and the customer's subscription. Moreover, to reach reactivity, the telecommunication operator needs to propose new offers with new services frequently. Considering those elements, and in order to optimize development and deployment costs, providers of network solutions use ASPL to implement the offers: stakeholders (i.e., marketing staff) are continuously involved and offers frequently changing (agility); they share the same bases (common components) and differ depending on the services proposed and the customer's subscription (variation points).

Another telecommunication market constraint concerns revenue problematic: a critical error generated after deploying an offer may cause important financial losses if not quickly fixed, depending on the volume of traffic and data transmission. That's why reactivity in tracing product generation path is very important.

With our approach, each generated product will have a marker composed from those of its components. Thus, we can easily identify the concerned elements to be checked and fixed. We can also identify the other impacted products and the related test cases to execute them and verify the product integrity (see Figure 3). When an offer is initiated by the management (based on market statistics and indicators, decision making system, etc.), it is implemented as a result of the instantiation of concerned components (use cases, design components, realization components and test scenarios) of the ASPL. Each component has its unique marker (UCi, DCi, RCi and Ti) and the generated product's marker (UC1, DC1, RC1, RC2, T1, T2, T3) is a result of their concatenation.

 ISBN: 978-1-61208-367-4

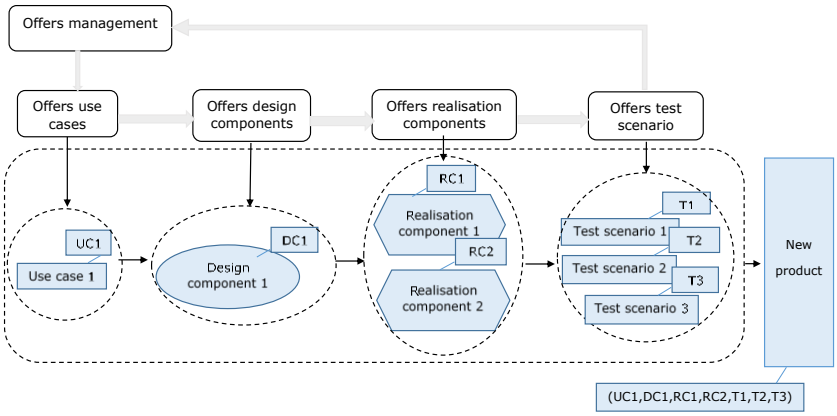


Figure 3. Simplified case study for telecommunication offers implementation

VI. CONCLUSION AND FUTURE WORK

Agile Product Line Engineering is a new promising method in software engineering. It helps companies gain flexibility, reactivity and customer satisfaction in a volatile and competitive context while optimizing costs and efforts.

We discussed in this paper the problematic of traceability in an ASPL through a state of the art, and proposed an approach for ASPL traceability based on markers and break-even points.

As the implementation of a break-even point requires a balance between the desired level of traceability and the costs of building and maintaining the agile system, our future contribution will focus on the optimization of the granularity and depth level of traceability in an ASPL.

REFERENCES

[1] L. M. Northrop, "SEI's software product line tenets, "IEEE Softw., vol. 19, no. 4, 2002, pp. 32–40.

[2] K. Pohl, G. Böckle, and F. Van Der Linden, Software product line engineering. 2005.

[3] J. Díaz, J. Pérez, P. P. Alarcón, and J. Garbajosa, "Agile Product Line Engineering - A Systematic Literature Review," Softw. Pract. Exp., vol. 41, no. 8, 2011, pp. 921–941.

[4] K. Tian and K. Cooper, "Agile and software product line methods: are they so different," 1st Int. Work. Agil. Prod. Line Eng. (APLE), collocated with 10th Int. Softw. Prod. Line Conf., 2006.

[5] Y. Ghanam, F. Maurer, and K. Cooper, "A Report on the XP Workshop on Agile Product Line Engineering," ACM SIGSOFT Softw. Eng. Notes, vol. 34, no. 5, 2009, pp. 25–27.

[6] R. Carbon, M. Lindvall, D. Muthig, and P. Costa, "Integrating Product Line Engineering and Agile Methods : Flexible Design Up-Front vs . Incremental Design," in 1st International Workshop on Agile Product Line Engineering APLE06, 2006, pp. 1–8.

[7] M. Fowler and J. Highsmith, "The agile manifesto," Softw. Dev., vol. 9, 2001, pp. 28–35.

[8] L. Northrop and P. Clements, "Software Product Lines," Carnegie Eng. Inst., 2005, pp. 1–105.

[9] C. Krueger, "Eliminating the adoption barrier," IEEE Softw., vol. 19, no. 4, Jul. 2002, pp. 29–31.

[10] M. a. Noor, R. Rabiser, and P. Grünbacher, "Agile product line planning: A collaborative approach and a case study," J. Syst. Softw., vol. 81, no. 6, Jun. 2008, pp. 868–882.

[11] S. Urli, M. Blay-Fornarino, P. Collet, and S. Mosser, "Using composite feature models to support agile software product line evolution," in Proceedings of the 6th International Workshop on Models and Evolution - ME '12, 2012, pp. 21–26.

[12] Y. Ghanam and F. Maurer, "Extreme Product Line Engineering : Managing Variability and Traceability via Executable Specifications," in 2009 Agile Conference, 2009, pp. 41–48.

[13] J. Cleland-Huang, O. Gotel, J. H. Hayes, P. Mäder, and A. Zisman, "Software Traceability: Trends and Future Directions," in ACM, 2014.

[14] N. Anquetil et al., "A model-driven traceability framework for software product lines," Softw. Syst. Model., vol. 9, 2010, pp. 427–451.

[15] Y. C. Cavalcanti et al., "Towards metamodel support for variability and traceability in software product lines," in Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems - VaMoS '11, 2011, pp. 49–57.

[16] N. Anquetil et al., "Traceability for Model Driven , Software Product Line Engineering 2 Software Product Line," ECMDA Traceability Work. Proc., 2008.

[17] K. Berg, J. Bishop, and D. Muthig, "Tracing software product line variability: from problem to solution space," Proc. 2005 Annu. Res. Conf. South Africain Inst. Comput. Sci. Inf. Technol. IT Res. Dev. Ctries., 2005, pp. 182–191.

[18] Y. Ghanam and F. Maurer, "An Iterative Model for Agile Product Line Engineering.," SPLC, 2008, pp. 377–384.

[19] B. G. K. Hanssen and T. E. Fægri, "Process fusion : an industrial case study on agile software product line engineering," J. Syst. Softw., 2008, pp. 843–854.

[20] P. O'Leary, F. M. Caffery, I. Richardson, and S. Thiel, "Towards agile product derivation in software product line engineering," 2009.

[21] P. O'Leary and F. McCaffery, "An agile process model for product derivation in software product line engineering," J. Softw. Evol. Process, 2012.

[22] M. Karam, S. Dascalu, and H. Safa, "A product-line architecture for web service-based visual composition of web applications," J. Syst. Softw., vol. 81, no. 6, 2008, pp. 855–867.

[23] J. Díaz, J. Pérez, and J. Garbajosa, "Agile product-line architecting in practice: A case study in smart grids," Inf. Softw. Technol., vol. 56, no. 7, Feb. 2014, pp. 727–748.

[24] M. Balbino, E. S. De Almeida, and S. Meira, "An Agile Scoping Process for Software Product Lines," in SEKE 2011 - Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering, 2011, pp. 717–722.

 ISBN: 978-1-61208-367-4

Implementing IT Service Management as an Organizational Change: Identifying Factors Affecting the Change Resistance

Marko Jäntti, Sanna Heikkinen
School of Computing
University of Eastern Finland
Email: {marko.jantti, sanna.heikkinen}@uef.fi

***Abstract*—When an IT organization decides to launch an IT service management programme, the discussion among employees after the decision is not always positive. New concepts, tools, roles and responsibilities and methods, as well as changes in strategy and culture may cause remarkable amount of change resistance among both employees and managers of a software company. Organizational change management is a fruitful research topic due to its challenging nature. It addresses employees' feelings, success factors of implementing strategic changes, role of change agents and factors affecting change resistance. In this study, we examine the IT service management programme as an organizational change and change resistance related to this change. Our research focuses on identifying symptoms of change resistance related to adoption of IT service management best practice framework IT Infrastructure Library (ITIL). The research problem of this study is: How does change resistance occur in IT service management programmes and how can it be decreased systematically? The main contribution of this paper is to present findings collected from five Finnish IT service provider organizations operating in different domains.**

Keywords*—IT service management; organizational change; change resistance.

I. INTRODUCTION

Transition from a software engineering company with software lifecycle models to a service oriented company with well-established IT Service Management (ITSM) processes and tools is a long journey that involves numerous challenges. Thousands of IT organizations world-wide use IT service management best practice framework IT Infrastructure Library (ITIL) in their service operations and globally, there are over million ITIL certified IT specialists. Despite the existence of ITSM best practices (ITIL [1], COBIT [2]), standards such as ISO/IEC 20000-1:2010 Part 1: Service management system requirements [3], maturity models and operational IT service management models, carrying out an IT service management programme successfully is a real challenge. The literature has shown that most major change initiatives (not only IT-related) seem to fail in the organizations. This has resulted in a need to examine how organizations can implement organizational changes successfully. There are strategic changes, such as changes in strategic technology platforms, changes related to service portfolio and changes in relationships between the service provider and strategic suppliers. IT changes, in turn, occur when somebody adds, modifies or removes IT services and their components. Often, there is no need to distinguish a strategic change from an organizational change because typically organizational changes are strategic changes.

One of the most famous persons related to the research field of organizational change is Kurt Lewin [4]. Lewin has proposed several important concepts of managing change such as action research, field theory, group dynamics and three step model of change. Lewin has presented a cyclic action research model that can be used to implement change. As a research method, action research emphasizes the role of actions taken during the research process. The method has been criticized with claims that it provides results that cannot be generalized to other organizations. Lewin applied a field theory for examining the human behavior. According to the field theory, the human behavior is related to both person and their environment. The concept of group dynamics refers to the Lewins finding that a group affects individual behavior while making decisions on organizational change.

In order to keep ITSM programme sustainable and live, one can apply organizational change models such as three step model of change. Three step model of change consists of three key stages: unfreezing, change and refreezing. During the first phase (unfreezing), one should ensure that the need for the change is understood in the organization and people desire the change. In other models, this is called creating urgency of change. During the second phase (change), a change is implemented by introducing, for example, creating new policies and procedures in the organization or communicating new values or attitudes to people working in the organization. In the third phase (refreezing), the focus is on reinforcing and supporting the change. The three step model of change addresses that organizational commitment must occur not only during the change but also before and after a change. In the ITIL framework, programmes that aim at continuous improvement of services are called Continuous Service Improvement Programmes (CSIP). CSIP is an ongoing formal programme undertaken within an organisation to identify and introduce measurable improvements within a specified work area or work process [5].

Only few academic studies have addressed the organizational change perspective in implementing IT service management programmes. The study of Tan, Cater-Steel and Toleman [6] emphasizes the role of the committed senior management in the ITSM projects success, as well as the role of a project champion and the recognition of the need for an appropri-

 ISBN: 978-1-61208-367-4

ate change management strategy. Additionally, in the study of Pollard and Cater-Steel [7] key success factors of ITIL implementations included top management support, training, virtual project teams, careful tool selection and use of external consults. Additionally, there are studies that have focused on conceptualising the ITSM projects [8]. The literature of organizational change management deals with the concept of change agents. Ford et al. [9] discuss how change agents may affect the resistance of change. They comment that change agents can participate in change resistance from their side by not legitimizing changes, misrepresenting changes and not calling for action (carrying out required actions). First, legitimization of changes refers to the action where change agents provide justification for changes, presenting clear rationale why the change is needed in the organization, and enhance the readiness for change adoption as well as increasing the likelihood of change acceptance. Legitimization of an ITSM programme might be, for example, a statement of required cost savings. Additionally, one should pay attention how fast change shall be adopted and what is extent of the change acceptance. The schedule plays an important role for example, when two IT service providers have agreed on merger by specified date.

Furthermore, Ford et al. [9], present that change resistance should be seen as a resource instead of a negative issue. Three types of value can be identified: existence value, engagement value, and strengthening value. Existence value means that change agents are able to keep conversation on change active and may also attract new members to participate in discussion. Conversation on ITSM programme may be started from service desk workers and then extended to other service areas. This conversation helps change agents to increase their understanding of the change. Engagement value can appear when change recipients engage in the change, for example, when they feel that the change will affect them negatively or when they are afraid of the organizations success. For example, a product manager engages in creating service catalogues because he/she feels that management does not understand the role of his / her product within the product portfolio. The strengthening value of resistance means that a conflict may strengthen recipients commitment on the change. A conflict could be, for example, a nature disaster or finding a common enemy, such as a tough competitor.

A. Our Contribution

The main contribution of this study is to study

- How change urgency related to ITSM programmes is communicated in the IT service provider organization?
- How IT service provider organizations introduce changes related to ITSM processes?
- How does change resistance occur in IT service management?
- Which methods are used by organizations to decrease the change resistance?

As results, this study provides new scientific knowledge on means how ITSM-related change urgency is communicated within an IT service provider, actions how organizational change is introduced, evidence on how change resistance occurs and methods how change resistance is decreased. The results of this study can be used by continual service improvement managers, ITSM programme managers, portfolio managers, service directors and other managers and team leaders responsible for introduction of ITSM processes. The remainder of the paper is organized as follows. In Section 2, the research methods of this study are described. In Section 3, the results of the study are presented. Section 4 is the analysis of findings. The discussion and the conclusions are given in Section 5.

II. Research Problem & Methodology

The research problem of this study is: How does change resistance occur in IT service management programmes and how can it be decreased systematically? The research problem was divided into the following research questions:

- How change urgency is communicated in the organization?
- How IT service provider organizations introduce changes related to ITSM processes?
- How does change resistance occur in IT service management?
- Which methods are used by organizations to decrease the change resistance?

A. Data Collection Methods

Data was collected from five IT service provider companies in Finland. Companies operated in various business domains (healthcare, energy, miscallenous services, bank and insurance). These organisations were selected for this study because they were representative cases of IT organisations with many year's experience in ITIL and ITSM. The following data collection methods/sources were used during the study:

- Documentation (ITSM process descriptions)
- Archives (incident records, change request records, service request records, email records)
- Interviews/discussions (service managers, directors, development manager, product/service managers)
- Participative observation (ITSM Awareness training periods in two organizations)
- Physical artefacts (Organization's intranet, ITSM tool)

We used case study research and action research [10] to collect the data on IT service management improvement programmes and service management process improvement. According to Yin [11], a case study is "an empirical inquiry that investigates a contemporary phenomenon within its real-life context". We used an exploratory case study with multiple case design. However, we decided to keep organizations and roles of respondents anoymous because it might be relatively easy to combine a person to a role and because organizational change management issues can be tricky and sensitive issues within the organization. Results of the case study resulted in change resistance-related information as by-product. Action research was used because suits well to the studies of organizational change. In fact, the roots of action research are in organizational change.

 ISBN: 978-1-61208-367-4

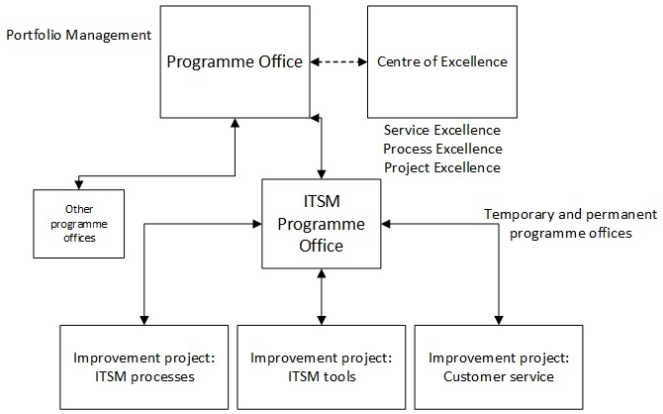


Fig. 1. Implementing ITSM Programmes

B. Data Analysis

The data of the case study were collected and analyzed using a within case analysis technique [12] that focuses on analyzing each case stand-alone before making any comparisons. Research findings were validated through discussions, seminars and workshops with the representatives of case organizations. In each case, the research team produced a case study report which was delivered to the case organization's contact persons. A qualitative content analysis technique was applied for case study material to build categories based on the comments from IT service provider organizations' managers and employees.

III. ITSM Programmes as Organizational Changes

In this study, we apply Lewins three step model of change and its phases (unfreezing, change and refreezing) to organizational changes where the context of change is adoption of IT service management process frameworks (IT Infrastructure Library) and service management standards. In unfreezing phase, we should first identify what is the rationale behind the change, what are the clear benefits of change, how to communicate the urgency of change to employees and managers of the IT organization and how to identify skilled and motivated change agents. Figure 1 shows an example of how ITSM programme could be implemented.

The change management model of Lewin is a very useful tool for organizing the findings from IT service provider organizations change efforts. During the study we observed that the change urgency related service management improvement is created in IT companies by addressing customers or owners need for a change or bottlenecks in the current IT infrastructure. The case organization's representatives reported following factors that triggered the adoption of ITSM frameworks in their organization:

- "Our key customer would like to improve service level management with us because they are not satisfied with the current practices."
- "We need defined service management processes because there have been plans on merging two organizations together."
- "The owners of the company encourage us to increase service quality."
- "IT customers are continuously requesting new types of IT service reports. Producing those manually takes a lot of time. The new tool should support effective measurement and reporting."
- "Customers are not interested in buying our new product because it lacks some specific features."
- "Our new management started a cost savings programme. We need measurable processes in order to improve, for example, the througtput times of processes."

A. How IT Service Provider Organizations Introduce Changes Related to ITSM Processes?

During the second phase (change) of the three step model of change, a change is implemented by introducing, for example, creating new policies and procedures in the organization or communicating new values or attitudes to people working in the organization. In our study, we observed that the IT service provider companies valued the following factors in introducing ITSM processes. Many companies in our study were implementing IT service change management processes.

- All the employees in the service area were provided a half day process training (TR)
- Motivating employees why change management is important (MT)
- A short general information package on ISO/IEC 20000 and ITIL Change Management (TR)
- Defining the scope of the process (P)
- Defining the processes in general level (P)
- Identifying the different types of changes: normal change, standard change, emergency change (P)
- Practicing identification of changes with practical examples (E)
- Publishing roles and responsibilities (RR)
- Starting the Change Advisory Board practice (F)
- Publishing the schedule, goals and monitoring (MN, G)
- Main focus should be in managing processes (P)
- Mapping the process improvement goals to organizational goals.(G)
- Meaningfullness of improvement work (ME)
- Ensuring that the right direction has been selected (MN)
- Consistent human resource management plays important role
- Proceeding in understandable steps (S)
- Integrated to annual goals (G)

 ISBN: 978-1-61208-367-4

- Tailoring ITSM to organization's own context (not just copy the processes from ITIL) (TA)
- Searching common good things (participative approach instead of orders) (PA)
- The goal should be to learn together and on individual level (L)
- Inspire yourself and others (MT)
- Finding the levels of freedom in developing processes (P)

B. How Does Change Resistance Occur in IT Service Management?

The following comments that address the change resistance were captured from our industrial partners:

- "Change resistance occurs in coffee table discussions"
- "Why should we change existing functioning practices?"
- "This approach is not suitable for us. It is too unflexible"
- "ITSM is yet another new ISM"
- "Does it make any sense to assign all the changes including the minor ones to the Change Advisory Board, it sounds very bureaucratic."
- "I am not interested in participating in service management training, I just try to make this work better."
- "I would say that the introduction of the new IT service management system is painful because employees do not have enough knowledge on IT service management processes."
- "We should introduce some new practices that promote the IT service management. For example, we could market the problem management process for application managers by launching a campaign: Start your day with a problem."
- "I am little bit worried about how processed and daily practices could be brought together."
- "Can you mention why should we learn these ITSM things if the managers only goal is to decrease the number of people?"

C. Methods for decreasing the change resistance related to ITSM

During the study we observed that organizations had used or expressed the need for the following methods:

- "Why are they continously asking that how could we improve things but they never tell us what has been improved?"
- "In order to ensure that employees do not go back to old practices, I needed to communicate the benefits of release management in a weekly meeting during two years. Sometimes, I felt a little bit frustrated, but at the end I think it was worth it."
- Reinforcing the new practices and create conditions for giving up from old practices
- Training key persons and ensuring that they are committed in change
- Training the personnel, for example providing them ITSM Awareness training
- Hiding the ITIL terminology to the background
- Creating justification for ITSM based on real service operations, not based on ITIL framework.

IV. ANALYSIS

The data analysis was carried out by using qualitative content analysis technique for the case study material. The purpose of the analysis was to create meaningful categories that can be exploited by future studies in the area of organizational change management. Regarding the first research question (How change urgency is communicated in the organization?), we identified that ITSM programmes are started because of key customers' requirements, management plans to outsource or merge service operations, the company's owners request to increase the quality, dissatisfied customers or due to establishment of cost savings programmes. The service improvement programme may receive more negative comments and change resistance if the reason behind the improvement is just to outsource services to other parties.

Regarding the second research question (How IT service provider organizations introduce changes related to ITSM processes), we coded the findings and identified 11 categories: Training (TR), Motivating (MT), Process (P), Examples (E), Roles and Responsibilities (RR), Function (F), Monitoring (MN), Meaningfulness of improvement work (ME), Goals (G), Scope (S), Tailoring (TA) and Learning (L). Our findings support earlier findings of Pollard and Cater-Steel [7] on success factors in ITIL implementations. Common issues with the study of Pollard and Cater-Steel were Process priority and Training. The study of Pollard and Cater-Steel [7] and Hochstein [13] also considered Top management support as a success factor. In our case, Motivating category could have been named as Top management category. Our Goals category corresponds to Relevant and realistic objectives category in the study of Stelzer and Mellis [14]. In this study, we did not use the category Service culture but some categories, for example, Learning could be part of the service culture. We also established categories that were not visible in previous studies such as the Function (based on a finding that service providers tend to create new organizational structures to facilitate the change) and Meaningfulness of improvement work (based on a finding that management should empower employees to carry out improvement).

Regarding the third research question (How does change resistance occur in IT service management?), we observed that change resistance occurs and is visible in daily events, such as in coffee table discussions, not only after seminars held by management. Employees may see the ITSM processes and practices too bureaucratic, time consuming and unflexible. The reason for this might be the large number of different types of records (incidents, problems, changes, releases) the support request may go through. In both training sessions and process

 ISBN: 978-1-61208-367-4

improvement discussions, we observed that employees were afraid of that a Change Advisory Board function would cause a bottleneck for processing small changes.

Regarding the fourth question (Which methods are used by organizations to decrease the change resistance?) we established the following categories:

- Better communication: "Why are they continously asking that how could we improve things but they never tell us what has been improved"
- Repeating the message: "In order to ensure that employees do not go back to old practices I needed to communicate the benefits of release management in a weekly meeting during two years. Sometimes I felt frustrated but at the end I think it was worth it."
- Reinforcing: "Reinforcing the new practices and create conditions for giving up from old practices."
- Training: "Training key persons and ensuring that they are committed in change; training the personnel, for example providing them ITSM Awareness training."
- Terminology: "Hiding the ITIL terminology to the background."
- Business-based justification: "Creating justification for ITSM based on real service operations, not based on ITIL framework."

Our analysis can be constructed into following recommendations how to communicate the ITSM programme as an organizational change to employees and managers: 1) analyze the received questions and comments on the ITSM programme and build clear counterarguments, 2) ensure that there is enough communication on ITSM programme (vision, goals, urgency) and that communication is done frequently, 3) use multiple channels (training, seminars, newsletters, social media) for communicating the benefits of ITSM programme but ensure that the core message is always consistent and clear, 4) explore whether change recipients (service personnel) have understood the ITSM programme-related communication correctly, 5) as a change agent use participative method to invite people for planning the change and learning together, 6) remember that talks do not necessarily lead to action. Thus, one should define a clear action plan for the ITSM programme. Action plan may cover improvements to processes, services or employees' competences.

V. Conclusion and Future Work

The research problem of this study is: How does change resistance occur in IT service management programmes and how can it be decreased systematically? We used case study methods to identify symptoms of IT service management - related change resistance. Data was collected from case studies, ITSM seminars, ITSM training sessions and ITSM action research periods with IT service provider organizations. Our results contribute to the research field of service science and organizational change. We identified triggers for establishment of ITSM programmes, studied how change resistance is visible in ITSM and how IT companies deal with the change resistance.

There are certain limitations related to our study. First, data were collected from five IT service provider organizations in Finland by using qualitative methods. Quantitative data collection and analysis could have provided new insights to the research topic. Second, we used qualitative content analysis only for case study material. The content analysis could have been used also for article material related to ITSM as an organizational change. Third, case study research method has received criticism that results from case studies cannot be generalized to other organizations. However, the research method literature indicates that they can still be used to improve and extend the theory. Further research could explore deeper the establishment of IT service management programmes and how change resistance has been taken into account in planning the programme.

Acknowledgment

We would like to thank the case organization's representatives for valuable feedback and responses that helped us to perform this study.

References

[1] Cabinet Office, *ITIL Service Strategy*. The Stationary Office, UK, 2011.

[2] COBIT 5, *Control Objectives for Information and related Technology: COBIT 5: Enabling Processes*. ISACA, 2012.

[3] ISO/IEC 20000:1, *Part 1: Service management system requirements*. ISO/IEC JTC 1 Secretariat, 2010.

[4] J. Helms-Mills, K. Dye, and A. J. Mills, *Understanding Organizational Change*. USA: Taylor & Francis, 2009.

[5] OGC, *ITIL Planning to Implement*. The Stationary Office, UK, 2002.

[6] W.-G. Tan, A. Cater-Steel, and M. Toleman, "Implementing IT service management: A case study focussing on critical success factors," *Journal of Computer Information Systems*, vol. 50, no. 2, 2009.

[7] C. Pollard and A. Cater-Steel, "Justifications, strategies, and critical success factors in successful itil implementations in u.s. and australian companies: An exploratory study," *Information Systems Management*, vol. 26, no. 2, pp. 164–175, 2009.

[8] J. Iden and T. R. Eikebrokk, "Understanding the ITIL implementation project: Conceptualization and measurements," in *Proceedings of 2011 22nd International Workshop on Database and Expert Systems Applications*. Washington, DC, USA: IEEE, 2011.

[9] J. Ford, L. Ford, and A. D'Amelio, "Resistance to change: The rest of the story," *Academy of Management Review*, vol. 33, no. 2, pp. 362–377, 2008.

[10] R. Baskerville, "Investigating information systems with action research," *Commun. AIS*, p. 4.

[11] R. Yin, *Case Study Research: Design and Methods*. Beverly Hills, CA: Sage Publishing, 1994.

[12] K. Eisenhardt, "Building theories from case study research," *Academy of Management Review*, vol. 14, pp. 532–550, 1989.

[13] A. Hochstein, G. Tamm, and W. Brenner, "Service oriented it management: Benefit, cost and success factors," in *Proceedings of the 2005 European Conference on Information Systems (ECIS 2005)*. AIS Electronic Library, 2005.

[14] D. Stelzer and W. Mellis, "Success factors of organizational change in software process improvement," *Software Process: Improvement and Practice*, vol. 4, no. 4, pp. 227–250, 1998.

 ISBN: 978-1-61208-367-4

Spider-PE: A Set of Support Tools to Software Process Enactment

SPEM Process Enactment in the CMMI-DEV and MR-MPS-SW Context

Carlos Portela, Alexandre Vasconcelos
Informatics Center (CIn)
Federal University of Pernambuco (UFPE)
Recife, Brazil
{csp3, amlv}@cin.ufpe.br

Sandro Oliveira, André Silva, Elder Silva
Postgraduate Program in Computer Science (PPGCC)
Federal University of Pará (UFPA)
Belém, Brazil
srbo@ufpa.br, {aandrecunhas, elderferreirass}@gmail.com

***Abstract*—In order to have a competitive software industry, it is essential to adopt standards and reference models of software processes quality. However, despite the growing adoption of standards and models, the number of organizations that adopt these is a small portion of the total population of software organizations. This paper presents a set of support tools to enactment of modeled processes in the Software Process Engineering Metamodel (SPEM). This set of support tools, called Spider-PE (Process Enactment), aims to assist software organizations in the implementation of the Capability Maturity Model Integration for Development (CMMI-DEV) and Reference Model of Software Process Improvement to Software (MR-MPS-SW) models. We expect Spider-PE to be more easily adopted by software organizations because it is based on models and standards largely accepted. Furthermore, this set of support tools adopts free technologies (non-proprietary) in order to reduce costs.**

Keywords-Software Process Enactment; Quality Models; SPEM; CMMI-DEV; MPS.BR.

I. INTRODUCTION

Achieving competitive advantage, to software companies, involves not only product's quality improvement and parallel services, but also the process production and software distribution [1]. In order to, nationally or internationally, have a competitive software department, it is essential the adoption of patterns, standards and reference models when it comes to processes. In this context, the Capability Maturity Model Integration for Development (CMMI-DEV) [2] stands out internationally speaking, while the Reference Model of SPI to Software (MR-MPS-SW) stands out in the Brazilian scope [1].

Although the adoption of norms and reference models for Software Process Improvement (SPI) is increasing, the amount of organizations that adopt these models is a small portion out of the total population of software organizations [3]. Different studies were conducted in order to understand why organizations do not adopt the standards and models for process improvement, and they indicate to questions regarding high costs, lack of support tools and bureaucracy related to the big amounts of resources demanded by the process execution [3][4]. Other studies were conducted in order to identify the critical success factors in software process improvement initiatives [5][6]. These studies take into account the use of support tools as critical success factor in a software process improvement program.

In this context, the purpose of this paper is to introduce the set of support tools Spider-PE (Process Enactment), which gives support to the flexible and semi-automated process execution, and that is adherent to the quality models CMMI-DEV and MR-MPS-SW. This set of tools is based upon free standards and technologies and is the outcome of the SPIDER Project (Software Process Improvement: DEvelopment and Research).

Besides this introductory part, Section II discusses the software process definition, execution and improvement steps, and it also talks about related work. In Section III, a set of tools Spider-PE and its components are presented. Section IV discusses the analysis of the SPEM 2.0 models' execution and the adherence to the CMMI-DEV and MR-MPS-SW models. Section V presents the results obtained in this research in both academy and industry. Finally, Section VI presents a conclusion.

II. BACKGROUND AND RELATED WORK

In this section, we present the main concepts and definitions of this research; then, we briefly describe the related work.

A. Concepts and Definitions

To represent the elements that integrate the process, or in other words, to build the software process models, it is necessary a language to model them [8]. One of the purposes of this type of representation is to facilitate the software process continuous improvement because it enables the understanding of the process in a visual and representative manner among the elements that compose the process [9].

In this domain, two approaches that have a large acceptation in the software industry regarding the modeling area were identified: Software Process Engineering Metamodel (SPEM) [10] and Business Process Modeling Notation (BPMN) [11]. The comparison between these two

 ISBN: 978-1-61208-367-4

standards and the justifications for the adoption of SPEM in this research are presented by Portela et al. [12].

After modeled, the process enters in the stage of execution, where it will be executed, controlled, validated and improved in short incremental and iterative cycles [13]. According to Reis [14], this execution phase depends on an automated mechanism that comprehends the modeled process and guides the developers during their work, as well as executes automatically some tasks.

In accordance with Oliveira et al. [15], one of the most important evolutions in the field of software quality is in the finding that the quality of the software product process is as important as the quality of the final product. From this affirmation, important software quality evaluation and certification mechanisms emerged based on the maturity and capability of the development organization in the conduction of their processes. Therefore, two improvement models are considered in this paper: the CMMI-DEV [2] and the MR-MPS-SW [1], which has been largely adopted in the Brazilian market.

B. Related Work

The WebAPSEE [16] environment allows the software process management. Based on free software, this environment uses its own visual language to model, the WebAPSEE-PML that is based in the formal specification defined in Reis [14]. França et al. [17] highlights the use of this environment in the adoption of the MPS.BR level G in an organization that develops software. However, there are no explicit evidences that the environment supports some of the results from this level, for example, the vertical traceability.

The Ontology-based software Development Environment (ODE) environment was designed based upon on a specific ontology for Software Quality [18]. The project ODE affirms that ADSs built based on anthologies allows an easier integration with different tools that aids software engineering activities.

The TABA station [19] aims to integrate support tools according to organization specificities, software processes and projects. The tool AvalPro supports the Process and Product Quality Assurance group from an organization. The tool Pilot supports the evaluation of improvement proposes of a process in a systematic, planned and controlled way. Using these tools, the TABA station presents an explicit support to the CMMI levels 2 e 3 e the MR-MPS-SW levels G to C processes areas.

The ImPPros environment [20] supports the implementation of software processes in an organization progressively. This approach takes into account the use of quality models and norms that guide the continuous process improvement and the software process transformation bases on the possible mapping among these models and standards.

Differently from these other environments, the approach presented in this paper introduces a execution formalism based on the standard SPEM 2.0, allowing the process model execution without using intermediary models, as it can be seen in details in Section IV. Furthermore, this approach considers the models CMMI-DEV and MR-MPS-SW through the utilization of good practices related to the institutionalization degree of the process execution in an organization that develops software.

III. SPIDER-PE: SET OF SUPPORTING TOOLS

The set of supporting concept adopted in this paper defines a set of technologies that can be integrated in order to aid in the software process execution. In this context, there are tools, techniques, procedures, processes, roles, methodologies, frameworks, languages, standards, patterns, and so on.

A. xSPIDER_ML: Enactment Language

Although the SPEM [10] is a standard defined by OMG [11], it does not offer native mechanisms to automated software process simulation and execution. Because of such limitation, a language for execution was defined, xSPIDER_ML. The xSPIDER_ML (eXecutable SPIDER_ML) is an extension of the modeling language SPIDER_ML [21], which is defined as a profile of SPEM that aids the process execution flexibly [22].

The xSPIDER_ML's structure was defined based on the structure proposed by xSPEM [13], since the both approaches have as goal turn the SPEM 2.0 into executable language. This structure divided in packages provides ways to define the conceptual structure to organizations, providing the necessary notions to execute their developing processes. Therefore, the xSPIDER_ML structure was divided in five packages: xSPIDERML_Core, ProcessParameters, ProjectVariables, EventTypes and ProcessTrace.

In order to be clear, only a subset of concepts of these packages will be presented, and they were selected in accordance with their relevance to the understanding of the components that composes the xSPIDER_ML. The components are available in the technical specification [23].

The execution operates on the instantiated processes and because of that, the elements of the SPIDER_ML are gathered in the package xSPIDERML_Core. Besides these, the package xSPIDERML_Core reuses the concepts and elements offered by the xSPEM [13] and the SPEM 2.0 [10] in order to provide all the necessary elements to define and organize a software process for later execution. These elements define the basis for all the remaining packages of the xSPIDERML.

In this package, there is the component Activity, a specialization of the WorkBreakdownElement and WorkDefinition, which defines basic work units in a process, as well as in a process per se. The class Process

 ISBN: 978-1-61208-367-4

represents a set of work definitions partially ordered with the intention of achieving the development goals, such as the delivery of a system. These processes are defined as sequences of Phases and Milestones, and they express the life cycle of a product being developed. Also in this package, there is the class TaskUse that represents the instance to a TaskDefinition (a class in the SPEM 2.0 structure and SPIDER_ML). This class must provide information related to the resources that will be involved during the execution of the task that it represents.

After the structure of the xSPIDER_ML's components be defined, it is necessary to define the rules that will be applied in these elements and their relationships. In this context, rules define ante and post conditions in the same way as an inference engine of an expert system [14]. These rules extend the SPIDER_ML's semantic and consequently the SPEM 2.0's semantic in order to represent the dynamic information inherent to the properties defined before. In order to be clear, it is presented in this section an example of these rules. The complete detailing of these basic rules is available in [23].

According to the xSPIDER_ML's structure, it is possible to identify a common aspect to the TaksUse components. One task can have a status notStarted, started, paused, finished. An abstract observation of the operational semantic of processes in execution related to this property can be accomplished. Considering t as a task to be executed and whose initial status is notStarted, the possible transitional relationships for t are presented in Figure 1.

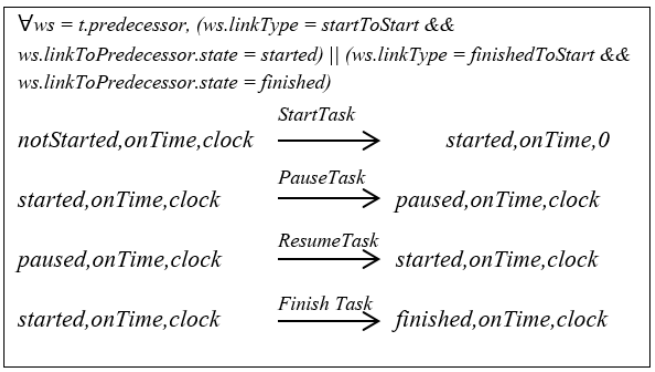


Figure 1. State Transition of a task *t*.

B. SPIDER_PE: Process Enactment Framework

The SPIDER-PE framework was defined in [31], and it aims to support the flexible execution of software processes adherently to the capability levels of the quality models CMMI-DEV and MR-MPS-SW.

It was decided to work with the capability dimension because it relates to the process execution definition directly. Process' capacity is the degree of refinement and institutionalization that the process runs in an organization/organizational unit [1].

The concept of framework adopted in this paper pictures the customization of a process to follow one or more recommendations of the quality models from the perspective of a generic activity flow that are necessary to execute any software process. Figure 2 presents three phases that compose this *framework*. To a complete description of this phases and its components; see [31].

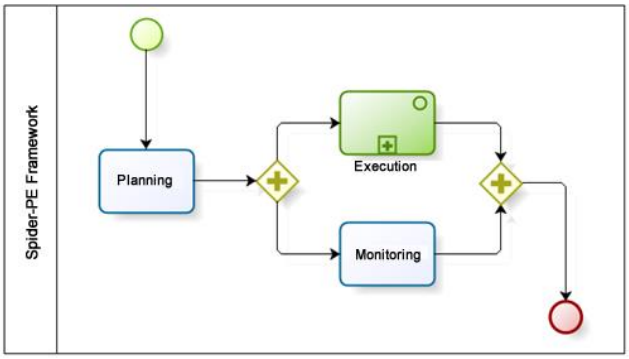


Figure 2. Phases of Enactment Process Framework.

The first phase of the framework is the Process Management Phase that has three steps: Planning, Execution and Monitoring. This phase has as goal the realization of the planning step and the accompaniment of the process execution according with the recommendations of the quality models CMMI-DEV and MR-MPS-SW. In this phase, the Process Policy is defined, the resources and the schedule are estimated, and so forth. Besides that, this phase has a sub phase called Execution of Process Activities, which is defined as the second phase of the framework.

Therefore, the next phase is the Execution of Process Activities, responsible for the actual process execution. In this phase, the team is responsible for creating the work products that are required to conduct the project's activities. To each work product generated, the Configuration Manager must perform the versioning and the control of the access. In the milestones and project's control points, the Quality Assurance team must verify the adherence of the process and work products to the standards and templates.

The third and final phase of the framework consists in the Application of the Execution Formalism, where the tool Spider-PE is responsible to apply the execution formalism defined in the technical specification of the xSPIDER_ML [23]. The purpose of this phase is enable the Framework SPIDER-PE to incorporate the rules and characteristics of the xSPIDER_ML that allows the activities to be instantiated by an execution machine and, consequently, meet the concept of flexibility and semi-automated execution specified in this language. Therefore, this phase must occur in parallel to the other phases of the framework, for this formalism must work on the activities of these phases.

 ISBN: 978-1-61208-367-4

C. Spider-PE: Process Enactment Tool

The Spider-PE is a GPL – *General Public License* [24] tool that brings a solution to the semi-automated software process execution [25]. This tool was created as a desktop environment using Java, and it was based on the use of free technologies [26], such as the IDE Eclipse 3.7, the SGBD MySQL 5.5, the objected-relational mapping *framework* Hibernate 4.0, the library for object serialization XML and vice-versa XStream, the library for creation and manipulation of PDF files iText and the library to draw diagrams based on the graph theory JGraph.

Beyond the free technologies used to develop the tool, other free software systems were used to aid the activities in the Spider-PE. The use of these tools allow the reuse of functionalities that meet the recommendations of the quality models CMMI-DEV and MR-MPS-SW, avoiding the development of specific functionalities to meet certain activities of the Framework SPIDER-PE. Hence, the following tools were integrated to the Spider-PE:

- Subversion (SVN), a system that adopts the Apache License of free software, and it is available in [27], to aid the changing management;
- Redmine, a GPL license tool, available in [28], and that allows the recording, monitoring, and accompaniment of the possible solutions to the different problems that can arise during the project execution;
- Spider-APF, that allows estimates to be made using the Function Point Analysis (FPA); Spider-UCP, that is used to measure the quantity of software from the perspective of the Use Case Points (UCP); and the Spider-CoCoMo, that allows time, effort and team quantity estimates to be using the CoCoMo method – Constructive Cost Model;
- Spider-CL, that aids the process and work product evaluation through objective criteria (described in evaluation checklists);
- Spider-MPlan, which supports the measurement process, which enables the definition, collection, analysis and monitoring measures.

The tools developed by SPIDER Project are under the GPL license and are available in [29].

The Spider-PE tool uses the concept of modules to specify a particular set of features, grouped according to the phases of the framework and the actors responsible for its implementation. Thus, this tool has three modules: Management, Process Management and Process Execution.

The first module to be accessed is the Directors module, which is responsible for setting the tools described in the Subsection 3.3.1. This module is also responsible for converting the XML file that contains the process modeling (built in Spider-PM [30] through notations of the SPIDER_ML language) to the relational database. The Administration module also allows the user to define a Process Manager, who will be responsible for managing, planning and monitoring the process.

Once defined, the Process Manager can access the Process Management module. This module is based on the adherence to the good practices in the quality models CMMI-DEV and MR-MPS-SW; therefore, it is related to the activities of planning and monitoring the process. The Process Management module consists of three phases: Planning, Execution and Monitoring. This module starts from planning the process, where the Process Manager has access to many features.

In the Execution Module Process, the human resources allocated to specific tasks may perform semi-automated and flexible process activities, as can be seen in Figure 3.

Figure 3. Enactment of a process task.

This module consists of the application of the execution formalism xSPIDER_ML. This application occurs in parallel to the steps of Process Execution and Monitoring in the Management module and because of this, the formalism runs concurrently with the activities of this module through the execution engine.

For a full description of the modules and each functionality; see the work of Silva et al. [25][26].

IV. SPIDER-PE EVALUATION

In this section, the set of support tools is evaluated from each of its components.

A. SPEM Models Enactment

The execution language xSPIDER_ML presents itself as a viable proposal for execution of process models defined in the standard SPEM 2.0. First, it is necessary to model the process in the Spider-PM [30] tool. This tool allows the modeling of processes using the notations of SPIDER_ML (profile SPEM 2.0). After the modeling stage, the Spider-PM allows saving an XML with the process modeling. The Administration module of the Spider-PE allows the user to export the information from the modeled process, saved in the XML file, to the relational database. By using the JGraph library, it is possible to apply the rules and formalisms implemented in the xSPIDER_ML under the SPEM notation, as shown in Figure 3.

In this image, the user selected the task Instantiated Task 03 to change its state (Started, Paused, or Finished). By selecting one of these states, the Spider-PE will be responsible for applying an internal mechanism of inference rules, and if there are no restrictions, the new information will be recorded on the database.

B. Adherence to CMMI-DEV and MR-MPS-SW

The tool Spider-PE provides support to the

 ISBN: 978-1-61208-367-4

implementation of Level 2 of CMMI-DEV and Level F of MR-MPS-SW, where the process is considered managed. The choice of these specific levels is due to the fact that they are initial levels and, therefore, tend to be more complex to implement [7].

Thus, for each component of the Spider-PE, it were identified Results of Process Attributes (RPA) of the MR-MPS-SW and Generic Practices (GP) of the CMMI-DEV that are supported by these components. To perform this analysis of adherence, first it was defined component levels of support of the Spider-PE related to the recommendations of these quality models:

- **Total:** the components of the Spider-PE fully support the systematization of the recommendations of a particular set of RPA and GP;
- **Partial:** the components of the Spider-PE partially support the systematization of the recommendations of a particular set of RPA and GP. I.E., these components do not meet all the recommendations of these models;
- **Not support:** the Spider-PE components do not support the systematization of a particular set of RPA and GP recommendations.

Figure 4 shows the relationship between the components of the Spider-PE, the MR-MPS-SW's RPA and CMMI-DEV's GP. This relationship was made from the analysis of the required requirements to meet the recommendations of these models by three experts (officially certified) in the implementation and evaluation of these quality models.

	xSPIDER_ML	Framework SPIDER-PE	Ferramenta Spider-PE
RAP 1 e GP 1.1	!	!	!
RAP 2 e GP 2.1; RAP 3 e GP 2.2; RAP 5 e GP 2.3; RAP 6 e GP 2.4; RAP 7 e GP 2.5; RAP 13 e GP 2.6; RAP 8 e GP 2.7; RAP 4 e GP 2.8; RAP 10 e GP 2.9; RAP 9 e GP 2.10; RAP 13 e GP 2.6	✗	✓	✓
RAP 11, RAP 12 e RAP 14	✗	✓	✓

Label: ✓ Total ! Partial ✗ Not Support

Figura 4. Adherence between the Spider-PE Components to CMMI-DEV and MR-MPS-SW.

To view the details of each of the recommendations of the components listed in the first column of table in Figure 4, it is necessary to consult the official guides of the MR-MPS-SW [1] and CMMI-DEV [2] models. A complete analysis of the adherence of Spider-PE, including the results, is available in [25][31].

V. OBTAINED RESULTS

In this section, the results obtained in this research are presented.

A. In Academy

An initial version of the proposal of this work has been published and presented during the WTDQS - Workshop of Theses and Dissertations in Software Quality [32]. This research is characterized as a subproject of the SPIDER Project, and it was accepted in the 2011/2012 cycle of the PBQP-SW (Brazilian Program of Software Quality and Productivity). In 2012, a comparative study of the patterns of SPEM and BPMN modeling and the proposed implementing xSPIDER_ML language was published in the JSEA - Journal of Software Engineering and Applications [12][22]. The Framework SPIDER-PE was the subject of a dissertation defended at the Informatics Center in the Federal University of Pernambuco (CIn/UFPE) [31]. The research related to the Spider-PE tool was also published in the Free Software Workshop [26] and was ranked among the "Best Papers" in this event, and in the VIII Annual Workshop of the MPS [25].

B. In Industry

The technologies presented in this article are used by the authors in consulting projects related to process improvement. First, the xSPIDER_ML, as well as the results of this phase of the research, were used by companies that are SPIDER project partners, such as the *FabSoft* and the *Pronto Digital*, both located in Belém city. Basically, the language aided on the steps of defining and monitoring the projects. On the other side, the activities of the Framework SPIDER-PE are widely adopted in the implementation of the Level 2 of CMMI-DEV and F of MR-MPS-SW in organizations in which the authors provide consulting, located at *Porto Digital* (Recife city) and *Farol Digital* (João Pessoa city). Finally, it is noteworthy that the last feature of the Spider-PE tool was released in November 2013. Nevertheless, requests for using the tool in consulting processes in a partnership agreement were made by the company *SWQuality* (based in Recife city and subsidiaries Maringá and Belém cities).

VI. CONCLUSION

The purpose of the Spider-PE is to support software development organizations, so they can run their processes flexibly and in a semi-automated way according to the notations of SPEM and the recommendations of the quality models CMMI - DEV and MR- MPS -SW. Therefore, the purpose of the tool is to facilitate the adoption of these standards and quality models for software development.

The set of tools also aims to help the software industry to achieve more satisfactory levels of discipline from the combination of patterns, models, procedures, tools, techniques and methods that help in implementing the process in an automated way that provides information about the progress of the project.

 ISBN: 978-1-61208-367-4

A strong point of this proposal is that the tool is totally free and allows the academic community and/or the industry to contribute to the evolution and improvement of the tool. However, the components of this tool must be customized according to the profile and characteristics of the organization that will use it. Moreover, this tool must be implemented in the organizational department responsible for the software development, requiring, therefore, a strategic, tactical and operational effort of the senior management so it can be deployed in a proper and satisfactory manner.

REFERENCES

[1] Association for Promotion of Brazilian Software Excellence. *SOFTEX: General Guide for Software Process Improvement*. Available from: http://www.softex.br/wp-content/uploads/2013/07/MPS.BR_Guia_Geral_Software_2012.pdf 2014.08.02

[2] Software Engineering Institute. *SEI: CMMI for Development*. Available from: http://www.sei.cmu.edu/reports/10tr033.pdf 2014.08.02

[3] M. Staples, et al. "An exploratory study of why organizations do not adopt CMMI", Journal of Systems and Software, vol. 80, pp. 883-895, Jun. 2007, doi:10.1016/j.jss.2006.09.008.

[4] G. Leal, et al. "Empirical study about the evaluation of the implantation of MPS.Br in enterprises of Paraná", Proc. Conferencia Latinoamericana En Informatica (CLEI 12), Oct. 2012, pp. 1-9, doi:10.1109/CLEI.2012.6427201.

[5] C. Almeida, T. Macedo, and A. Albuquerque. "Analysis of the continuity of software processes execution in software organizations assessed in MPS.BR using Grounded Theory", Proc. International Conference on Software Engineering & Knowledge Engineering (SEKE 11), Jul. 2011, pp. 792-797.

[6] M. Montoni and A. R. Rocha, "An Investigation into the Success Critical Factors in Initiatives Software Process Improvement". Proc. Brazilian Symposium on Software Quality (SBQS 11), Jun. 2011, pp. 151-165.

[7] S. Oliveira, et al. "A Proposal for Systemic Solution of a Free Software Tools SUITE to Support to MPS.BR Model Implementation", Magazine of Brazilian Program of Quality and Productivity in Software, 2nd ed., pp. 103-107, 2010.

[8] H. Feiler and S. Humphrey, "Software Process Development and Enactment: Concepts and Definitions", Proc. International Conference on the Software Processes - IEEE Computer Society Press, Feb. 1993, pp. 28-40, doi: 10.1109/SPCON.1993.236824.

[9] M. Kellner and G. Hansen, Software Process Modeling. Pittsburgh, PA: Carnegie Mellon University/Software Engineering Institute, 1988.

[10] Object Management Group. *OMG: Software & Systems Process Engineering Meta-Model Specification*. Available from: http://www.omg.org/spec/SPEM/2.0/PDF 2014.08.02

[11] Object Management Group. *OMG: Business Process Model and Notation (BPMN)*. Available from: http://www.omg.org/spec/BPMN/2.0/PDF 2014.08.02

[12] C. Portela, et al. "A Comparative Analysis between BPMN and SPEM Modeling Standards in the Software Processes Context", Journal of Software Engineering and Applications, vol. 5, pp. 330-339, May 2012, doi:10.4236/jsea.2012.55039.

[13] R. Bendraou, B. Combemale, X. Cregut, and M.-P. Gervais, "Definition of an Executable SPEM 2.0", Proc. Asia-Pacific Software Engineering Conference (APSEC 07), Dec. 2007, pp. 390-397, doi: 10.1109/ASPEC.2007.60.

[14] C. Reis, "A Flexible Approach for Evolutive Software Process Enactment", PhD Thesis in Science Computer. Porto Alegre, RGS: Informatics Institute, Federal University of Rio Grande do Sul, 2003.

[15] S. Oliveira, A. Vasconcelos, and A. C. Rouiller. "A Proposal of Environmental to Software Process Implementation", INFOCOMP - Journal of Computer Science, vol. 4, pp. 71-78, Mar. 2005.

[16] A. Lima, et al. "WebAPSEE: A Free and Flexible Environment for Software Process Management", Proc. Free Software Workshop (WSL 06), Apr. 2006.

[17] B. França, E. Sales, C. Reis, and R. Reis. "Using the WebAPSEE Environment in the MPS.BR Level G implementation in CTIC-UFPA," Proc. Brazilian Symposium on Software Quality (SBQS 09), Jun. 2009, pp. 310-317.

[18] G. Guizzardi, R. Falbo, and R. Guizzardi. "Grounding Software Domain Ontologies in the Unified Foundational Ontology (UFO): The Case of the ODE Software Process Ontology", Proc. Conferencia Iberoamericana de Software Engineering (CIbSE 08), Feb. 2008, pp. 127-140.

[19] G. Travassos, "The Model Integration of TABA Station Tools", PhD Thesis in Science Computer. Rio de Janeiro, RJ: COPPE, Federal University of Rio de Janeiro, 1998.

[20] S. Oliveira, "ProDefiner: A Progressive Approach to the Definition of Software Processes in the Context of a Centered Environment in Process", PhD Thesis in Science Computer. Recife, PE: Informatics Center, Federal University of Pernambuco, 2007.

[21] R. Barros and S. Oliveira, "SPIDER_ML: A Language for Software Process Modeling", Proc. Regional School of Informatics (ERIN 10), Oct. 2010.

[22] C. Portela, et al. "xSPIDER_ML: Proposal of a Software Processes Enactment Language Compliant with SPEM 2.0", Journal of Software Engineering and Applications, vol. 5, pp. 375-384, Jun. 2012, doi:10.4236/jsea.2012.56044.

[23] C. Portela and M. Gomes. *xSPIDER_ML. Technical Specification.* Available from: http://www.spider.ufpa.br/projetos/xspider_ml/xSPIDER_ML.pdf 2014.08.05

[24] GNU Project. *General Public License*. Available from: http://www.gnu.org 2014.08.05

[25] A. Silva, E. Silva, C. Portela, S. Oliveira, and A. Vassconcelos. "Spider-PE: A Support Tool to Implementation of Capacity of the MR-MPS Level F and CMMI-DEV Level 2", Proc. Annual Workshop of the MPS.BR (WAMPS 12), Oct. 2012, pp. 186-194.

[26] A. Silva, E. Silva, C. Portela, S. Oliveira, and A. Vassconcelos. "Spider-PE: A Support Tool to Software Process Enactment adhering to CMMI-DEV and MR-MPS", Proc. Free Software Workshop (WSL 12), Jul. 2012.

[27] Apache Subversion. *SVN Download*. Available from: http://subversion.apache.org/ 2014.08.08

[28] Redmine Project. *Redmine Download*. Available from: http://www.redmine.org/projects/redmine/wiki/Download 2014.08.08

[29] SPIDER Project. *Research Results*. Available from: spider.ufpa.br/index.php?id=resultados 2014.08.08

[30] R. Barros and S. Oliveira, "Spider-PM: A Support Tool for Software Process Modeling", Proc. Annual Meeting of Computing (ENACOMP 10), Oct. 2010.

[31] C. Portela, "Spider-PE: A Support Framework to Flexible Enactment of Software Processes adhering to Quality Models", Master Dissertation in Science Computer. Recife, PE: Informatics Center, Federal University of Pernambuco, 2012.

[32] C. Portela, A. Vasconcelos, and S. Oliveira. "Spider-PE: A Support Set of Tools to Process Enactment adhering to Quality Models", Proc. Thesis and Dissertation Workshop on Software Quality (WTDQS 11), Jun. 2011.

 ISBN: 978-1-61208-367-4

On the Use of Ontology for Dynamic Reconfiguring Software Product Line Products

Thyago Tenório
Computing Institute
Federal University of Alagoas (UFAL)
Maceió, Brazil
ttmo@ic.ufal.br

Diego Dermeval
Department of Computer and Systems
Federal University of Campina Grande (UFCG)
Campina Grande, Brazil
diegodermeval@copin.ufcg.edu.br

Ig Ibert Bittencourt
Computing Institute
Federal University of Alagoas (UFAL)
Maceió, Brazil
ig.ibert@ic.ufal.br

***Abstract*—Software Product Line (SPL) is a set of software systems that have a particular set of common features and that satisfy the needs of a particular market segment or mission. The traditional SPLs focus on building software platforms at development time. In contrast, modern systems of emerging domains (e.g., ubiquitous computing, service robotics and autonomic systems) require new settings to perform dynamic reconfiguration. In this context, Dynamic Software Product Line (DSPL) extends the SPL concept to provide an efficient way to deal with software adaptation at runtime. A key artifact in SPL is the feature model. Such model is very important in the specification of SPLs, representing the variability of the software and also supporting the instantiation of applications. However, this model has some limitations regarding its usage in DSPL. In order to effectively provide dynamic reconfiguration of features, it is necessary to represent such model in a formal way thus it can be automatically monitored, retrieved and modified during the execution of a product. Hence, we propose an ontology for feature modeling, regarding its capabilities to handle changes in the feature models, demanding less effort to be reconfigurable at runtime. In order to illustrate the use of the ontology, a set of reconfiguration scenarios in the domain of ubiquitous computing are presented.**

Keywords–Ontology; Software Product Lines; Dynamic Software Product Lines;

I. INTRODUCTION

Software Product Line (SPL) engineering is a paradigm that advocates the reusability of software artifacts and the rapid development of new applications for a particular domain. These objectives are achieved by capturing the commonalities and variabilities between the products from the same domain in variability models (e.g., feature models). Software Product Line engineering methods offer characteristics such as rapid product development, reduced time-to-market, quality improvement, and more affordable development costs [1].

The traditional methods for designing SPL focus on its construction at development time, thus each product configuration is instantiated before a product is delivered to the customer. However, the modern systems of emerging domains such as ubiquitous computing, service robotics, unmanned space and autonomic systems are increasingly requiring new mechanisms capable to reconfigure their variability models at runtime, i.e., without stopping the system's execution. In this context, Dynamic Software Product Lines (DSPL) extend existing Software Product Line engineering approaches to provide ways to handle with software adaptation at runtime [2].

One of the key artifacts used in SPL engineering is the feature model. Such model is widely used in the context of SPLs to capture the common and variable functionalities of products from a same domain. However, its informal representation has several limitations regarding its usage in DSPLs, for instance, it is difficult to automatically monitor, retrieve and modify them at runtime [2].

In order to effectively provide dynamic reconfiguration of products, it is necessary to represent feature models in a formal way, as a result it can be automatically reasoned or queried during the execution of a product. Meanwhile, some studies use ontologies as an effective way to formally represent feature models [3][4]. However, none of the existing studies on ontology-based feature modeling provides explicit elements (e.g., status of the features and product configuration model) capable to allow product reconfiguration at runtime with less effort.

Facing the potential benefits of using ontologies to represent feature models for DSPL purposes and the limitations of existing ontology-based feature modeling approaches regarding dynamic reconfigurations, we propose the OntoSPL ontology. Such ontology presents an alternative way for modeling ontology-based feature models. OntoSPL was conceived with the purpose to be as much flexible as possible, since it specifies a predefined structure of classes and properties and suggests the creation of features model as OWL instances/individuals of such structure. In addition, we present a set of SPARQL queries, in different scenarios, that can be executed to automatically reconfigure SPL products specified in OntoSPL.

The remainder of this paper is organized as follows. Section II describes in details the OntoSPL ontology. Section III presents the OntoSPL for DPSL Products and a set of SPARQL queries for reconfiguring SPL products. Section IV compares our work to related ones. Finally, Section V presents our conclusions and points out future works.

II. ONTOSPL

This section presents the OntoSPL ontology. According to the Guarino's ontologies classification [5], such ontology is a domain one, which aims to describe the main concept of Software Product Line through the feature model diagram. The variability of SPLs is commonly expressed through features represented in this model. A feature is a property of the system which is relevant to some stakeholder and is used to capture similarities and variabilities in software systems.

Feature modeling has been proposed as an approach for

 ISBN: 978-1-61208-367-4

describing variable requirements for Software Product Lines [6]. It is an important activity of the Software Product Line development process, since it is in such phase that the common and variable features of the product family are specified. In this sense, OntoSPL provides an explicit conceptualization of the essential elements involved in such diagram and is described by the following elements: concepts, properties and relations. In the sequel, the ontology is informally described (through the description of feature model elements) and its concepts, properties and relationships are further presented.

A. *Informal description of the ontology*

The ontology is inserted in the SPL Domain Engineering. It describes the concepts involved in feature modeling, as proposed by the Feature-Oriented Domain Analysis (FODA) notation [7]. A feature model provides a graphical tree-like notation that shows the hierarchical organization of features. The root of the tree represents the whole SPL node, all other nodes represent different types of features which are part of a SPL.

Features are organized in feature models and can be one the following types: mandatory, optional, alternative and or-features. The mandatory type must be present in all products derived from a Software Product Line. The optional one may or may not be included into a product derived from a SPL, hence its presence is optional. In the alternative feature, exactly one feature from a set of features must be included in a product. In the or-feature type, one or more features from a set of features can be included in a product from a SPL. Moreover, dependency rules between features may exist and can be of two types: (i) Requires, when one feature requires the existence of another feature (they are interdependent); and (ii) Excludes, when one feature is mutually exclusive to another one (they can not coexist). The Group element indicates a constraint in a set of grouped features.

A feature constraint has also a name and can be classified in Depend (Require), Exclude or Group. The Depend constraint has a name, a set of source features and a set of target features. Such constraint means that if all source features are selected in a product, then, in the same product derived from a SPL, all the target features must be selected too. The Exclude constraint has exactly the same properties of the Depend one. It only has a semantic difference, since if all source features are selected in a product then any target features may not be selected in such product. Finally, the Group constraint has a name, a set of features and a constraint type which indicates a type of the constraint on the group.

B. *Description of the classes, properties and relationships*

In this section, the classes, properties and relationships of the OntoSPL are described. Figure 1 illustrates its hierarchy of classes. Note that the description of the classes on below follows the format ”Class_Name(Class_Attribute_1, Class_Attribute_2,...,Class_Attribute_n)”.

- *SoftwareProductLine* (name, description, FeatureModel): this class represents an arbitrary Software Product Line. It has primitive elements such as: name and description. Moreover, a SPL contains a Feature Model.
- *FeatureModel* (name, Feature, FeatureConstraint): this class describes a Feature Model which represents the hierarchy organization of the features of a SPL. It has a set of features and a set of feature's constraints.
- *Feature* (name, current_state): this class represents a resource available in the Software Product Line. It may be classified into Mandatory, Optional, Alternative, OrFeature and RootFeature:
  - *Mandatory* (name): this class represents a mandatory resource of the SPL, i.e., it must be present in all products
  - *Optional* (name): this class represents an optional resource of the SPL, i.e., it is optionally present in any product.
  - *Alternative* (name, exclusive, AlternativeFeature): this class represents an alternative resource of the SPL. An alternative resource specifies that two or more resources may not co-exist.
  - *OrFeature* (name, AlternativeFeature): this class represents an or exclusive resource of the SPL. An or exclusive resource specifies that two or more resources may or may not co-exist.
  - *RootFeature* (name): this class represents a root feature. A root feature represents the root of the features tree. It is on top of a feature model.
- *FeatureConstraint* (name): this class represents a constraint in the feature model. It may be classified into Depend, Exclude or Group:
  - *Depend* (name, SourceFeature, TargetFeature): This class represents a constraint of the Depend type. As mentioned above, it has a set of source features and a set of target features.
  - *Exclude* (name, SourceFeature, TargetFeature): this class represents a constraint of the Exclude type. As mentioned above, it has a set of source features and a set of target features.
  - *Group* (name, SetFeatures, typeConstraint): this class represents a constraint of the Group type. It has a set of features and a String typeConstraint which indicates the type of the

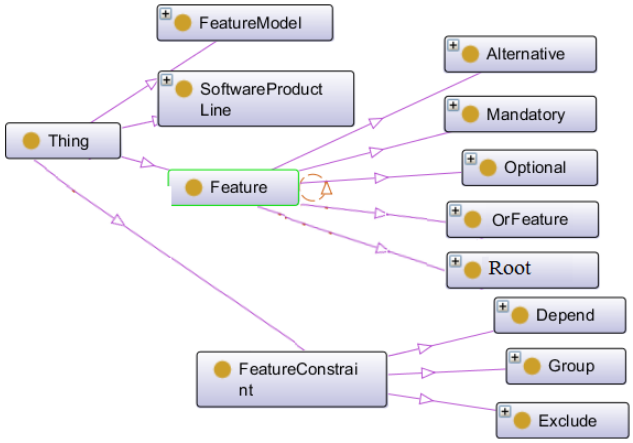


Figure 1. OntoSPL classes hierarchy (OntoViz plugin visualization).

constraint. The types can be: (i) zero-or-one feature exactly (0 or 1); (ii) At-least-one feature (1 or more); (iii) Exactly-one feature (1); (iv) Any feature (0 or more); (v) All features (n);

OntoSPL specifies a set of relationships between the ontology classes. The two classes between the parentheses following the property name represents, respectively, the source and target classes of such property.

- *hasRootFeatures* (FeatureModel, RootFeature): specifies that a FeatureModel contains a set of root features (which may not be empty);
- *hasSetOfAlternativeFeatures* (Alternative, Alternative): specifies that an alternative feature must have at least one feature alternative. It is a symmetric property;
- *hasSetOfConstraints* (FeatureModel, FeatureConstraint): specifies that a FeatureModel contains a set of feature's constraints;
- *hasSetOfFeatures* (Group, Feature): specifies that a Group constraint contains a set of features (which may not be empty);
- *hasSourceFeatures* (Depend/Exclude, Feature): specifies that a Depend or Exclude constraint has a set of source features (which must have at least one feature);
- *hasTargetFeatures* (Depend/Exclude, Feature): specifies that a Depend or Exclude constraint has a set of target features (which must have at least one feature);
- *isBasedOn* (SoftwareProductLine, FeatureModel): specifies that a SPL is based on exactly one FeatureModel. It is a functional property;
- *isChildOf* (Feature, Feature): specifies that a feature is child of exactly one another Feature. It is a functional property and it is also the inverse property of isParentOf;
- *isParentOf* (Feature, Feature): specifies that a feature contains a set of children features. It is the inverse property of isChildOf.

C. Axioms of the ontology

The classes and relationships described above express a taxonomy of the OntoSPL ontology. In order to describe it in a detailed and formal way, it must be governed with axioms. All axioms of the OntoSPL are defined in description logics (DL) and the OWL syntax used to represent it is summarized in Table I. The data properties, classes axioms and object properties are, respectively, presented in Tables II, III and IV.

TABLE I. SUMMARY OF DL SYNTAX.

Notations	Explanation
$\top$	Superclass of all OWL classes
A $\sqsubseteq$ B	A is a subclass of B
A $\sqsubseteq \neg$ B	A and B are disjoint class
A $\sqcap$ B	Class intersection
A $\sqcup$ B	Class union
A $\equiv$ B	Class equivalence
$\top \sqsubseteq \forall$ P.A	Range of property is class A
$\exists$ / $\forall$ P.A	allValuesFrom/someValuesFrom restriction that for every instance of this class that has instances of property P, all some of the values of the property are members of the class A

TABLE II. DATA PROPERTIES AXIOMS OF ONTOSPL.

Data Property	Source	Data Type
description	SoftwareProductLine	String
name	SoftwareProductLine; FeatureModel;Feature; FeatureConstraint	String
exclusive	Alternative	String
typeConstraint	Group	String
currentState	Feature	Boolean

TABLE III. CLASSES AXIOMS OF ONTOSPL.

Class	**Axioms**
Alternative	$Alternative \sqsubseteq Feature$ $Alternative \sqsubseteq \neg Optional$ $Alternative \sqsubseteq \neg Mandatory$ $Alternative \sqsubseteq \neg ORFeature$ $Alternative \sqsubseteq \neg Root$
Depend	$Depend \sqsubseteq FeatureConstraint$ $Depend \sqsubseteq \neg Exclude$ $Depend \sqsubseteq \neg Group$
Exclude	$Exclude \sqsubseteq FeatureConstraint$ $Exclude \sqsubseteq \neg Depend$ $Exclude \sqsubseteq \neg Group$
Feature	$Feature \sqsubseteq \top$
Feature Constraint	$FeatureConstraint \sqsubseteq \top$
Feature Model	$FeatureModel \sqsubseteq \top$
Group	$Group \sqsubseteq FeatureConstraint$ $Group \sqsubseteq \neg Depend$ $Group \sqsubseteq \neg Exclude$
Mandatory	$Mandatory \sqsubseteq Feature$ $Mandatory \sqsubseteq \neg Optional$ $Mandatory \sqsubseteq \neg Alternative$ $Mandatory \sqsubseteq \neg ORFeature$ $Mandatory \sqsubseteq \neg Root$
Optional	$Optional \sqsubseteq Feature$ $Optional \sqsubseteq \neg Mandatory$ $Optional \sqsubseteq \neg Alternative$ $Optional \sqsubseteq \neg ORFeature$ $Optional \sqsubseteq \neg Root$
ORFeature	$ORFeature \sqsubseteq Feature$ $ORFeature \sqsubseteq \neg Mandatory$ $ORFeature \sqsubseteq \neg Alternative$ $ORFeature \sqsubseteq \neg Optional$ $ORFeature \sqsubseteq \neg Root$
Root	$Root \sqsubseteq Feature$ $Root \sqsubseteq \neg Mandatory$ $Root \sqsubseteq \neg Alternative$ $Root \sqsubseteq \neg ORFeature$ $Root \sqsubseteq \neg Optional$
Software Product Line	$SoftwareProductLine \sqsubseteq \top$

III. DSPL PRODUCTS RECONFIGURATION

In SPL engineering, the applications are built by reusing domain artifacts and by exploiting the product line variability. To create instances (products) of a Software Product Line, one must choose the features that will be present in the product, following the constraints of the features model.

A SPL product contains a specific subset of features. In the DSPL context, by contrast, the requirements (expressed as feature models) of these products vary at runtime. Thus, these models must be reasoned or queried during the execution of a product, allowing it to be self-reconfigured at runtime, as required by DSPL engineering.

In this sense, OntoSPL may represent a single product based on a SPL feature model specified. It is used as the decision model on where the SPL variations are selected. Hereafter, we present how to represent a product on the ontology. Then, we present a set of dynamic reconfiguration scenarios which were applied to a running example of the

 ISBN: 978-1-61208-367-4

TABLE IV. OBJECT PROPERTIES AXIOMS OF ONTOSPL.

Object Property	Axioms
hasRootFeatures	$\exists$ hasRootFeatures Thing $\sqsubseteq$ FeatureModel $\top \sqsubseteq \forall$ hasRootFeatures ($\exists$ hasRootFeatures RootFeature)
hasSetOfAlternativeFeatures	hasSetOfAlternativeFeatures $\equiv$ hasSetOfAlternativeFeatures$^-$ $\exists$ hasSetOfAlternativeFeatures Thing $\sqsubseteq$ Alternative $\top \sqsubseteq \forall$ hasSetOfAlternativeFeatures ($\geq$ 1 hasSetOfAlternativeFeatures Alternative)
hasSetOfConstraints	$\exists$ hasSetOfConstraints Thing $\sqsubseteq$ FeatureModel $\top \sqsubseteq \forall$ hasSetOfConstraints ($\forall$ hasSetOfConstraints (Depend $\sqcup$ Exclude $\sqcup$ Group))
hasSetOfFeatures	$\exists$ hasSetOfFeatures Thing $\sqsubseteq$ Group $\top \sqsubseteq \forall$ hasSetOfFeatures ($\exists$ hasSetOfFeatures Feature)
hasSourceFeatures	$\exists$ hasSourceFeatures Thing $\sqsubseteq$ Depend $\exists$ hasSourceFeatures Thing $\sqsubseteq$ Exclude $\top \sqsubseteq \forall$ hasSourceFeatures ($\geq$ 1 hasSourceFeatures Feature)
hasTargetFeatures	$\exists$ hasTargetFeatures Thing $\sqsubseteq$ Depend $\exists$ hasTargetFeatures Thing $\sqsubseteq$ Exclude $\top \sqsubseteq \forall$ hasTargetFeatures ($\geq$ 1 hasTargetFeatures Feature)
isBasedOn	$\top \sqsubseteq \leq$ 1 isBasedOn Thing $\exists$ isBasedOn Thing $\sqsubseteq$ SoftwareProductLine $\top \sqsubseteq \forall$ isBasedOn (= isBasedOn FeatureModel)
isChildOf	isChildOf $\equiv$ isParentOf $^-$ $\top \sqsubseteq \leq$ 1 isChildOf Thing $\exists$ isChildOf Thing $\sqsubseteq$ Feature $\top \sqsubseteq \forall$ isChildOf (= isChildOf Feature)
isParentOf	isChildOf $\equiv$ isParentOf$^-$ $\top \sqsubseteq \leq$ 1 isParentOf$^-$ Thing $\exists$ isParentOf Thing $\sqsubseteq$ Feature $\top \sqsubseteq \forall$ isParentOf Feature

ubiquitous domain published in the literature, a simplified smart hotel [8].

A. DPSL products configuration

OntoSPL supports the instantiation of products based on the SPL in order to facilitate the reconfiguration of the product when it is necessary. In this sense, the property $current\_state$ of the Feature class indicates whether the feature belongs or not to a particular product. This property presents the following range of values: $\{"eliminated" : string, "selected" : string\}$. Such a property can only receive the values: selected, case the feature must be in the product, or eliminated, case the feature must not be in the product.

Hence, one can reason in the ontology to perform dynamic reconfiguration in an arbitrary product. After defining the features that may be present in the product to be created, there is only necessary to set the property $current\_state$ for each feature instantiated in a product.

For instance, Figure 2 depicts the feature model of the Simplified Smart Hotel (extracted from [8]). Its mandatory features are represented by a small filled circle above the feature name (e.g., *Automated Illumination*). Optional features are represented by a small circle not filled (e.g., *Piped Music*, *Security* and *Alarm*). Alternative features share the same parent feature and are graphically represented by a not filled arc below the parent feature; such arc means that one and only one of the child features must be chosen (e.g., *Silent Alarm*,*Siren* and *Visual Alarm*). Finally, the or-features (e.g., *Infrared Sensor* and *Volumetric Sensor*) are represented by a filled arc, in a similar way to alternative features.

As shown in Figure 2, the gray features indicate which are the selected features for a product configuration. The current configuration of the simplified smart hotel includes the *Piped Music*, *Security*, *In Room Detection*, *Volumetric Sensor*, *Alarm*, *Silent Alarm*, *Automated Illumination features* and *Lighting by Occupancy* features. However, the white features represent potential variants of a product configuration [8].

B. Dynamic reconfiguration scenarios

Once a product is specified in the OntoSPL ontology, it can be reconfigured dynamically according to different scenarios. This section describes scenarios directly related to changes in the type of a feature and also changes related to product configuration.

In this sense, to specify this product in the OntoSPL ontology, one must set the *current_state* property of these features to the "selected" value and "eliminated" for the others features. For instance, in our running example, the gray features of Figure 2 assume the "selected" value in the OntoSPL, whereas the white features assume the "elimated" value.

In the sequel, we present three scenarios (specified in SPARQL 1.1 [9]) which can be executed for changing SPL products at runtime. Note that these scenarios are presented by applying them to our running example, but they have generic purposes.

1) Changing an optional feature to mandatory feature: Changing an optional feature to a mandatory one demands a simple change scenario in the requirements of a particular feature. Such change does not have a great impact in the feature model, since it is not necessary to create/remove features on the feature model.

Let's suppose, for an arbitrary reason, that an Smart Hotel requires security requirements. In this context, there is the need to make the "Security" feature a mandatory one. This way, the query on Figure 3 could be used to perform such reconfiguration.

This query updates the property *type* of the feature being changed. Note that, the optional type is deleted while the mandatory type is inserted. As consequence, the feature will be mandatory in the product.

2) Selecting an optional feature in a product: This is one of the most common changing scenarios in SPLs. Usually, a product is firstly generated according to customer's needs at

 ISBN: 978-1-61208-367-4

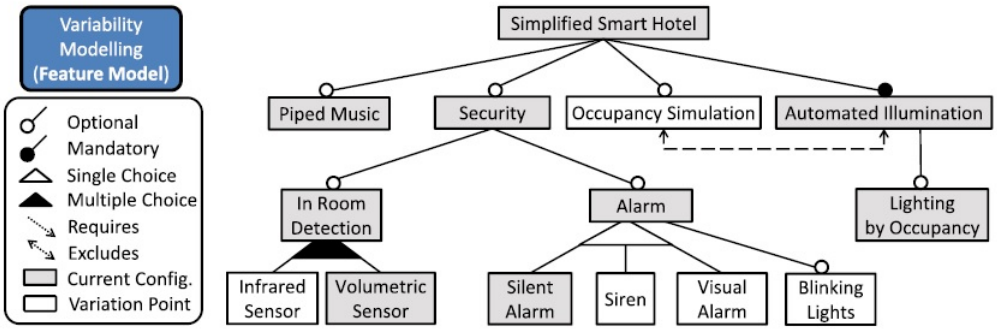


Figure 2. Feature model of the simplified smart hotel (extracted from [8]).

```
PREFIX spl:<http://nees.com.br/ontologies/SPL.owl#>
PREFIX rdf:<http://w3.org/1999/02/22-rdf-syntax-ns#>
DELETE {?x rdf:type spl:Optional.}
INSERT {?x rdf:type spl:Mandatory.}
WHERE{?x spl:name "Security".}
```

Figure 3. Changing an Optional feature to Mandatory feature.

development time and includes several features and, in such moment, a optional feature would not be selected to be in the product. However, in a later moment, it is possible that a client demands the inclusion of an optional feature that was not previously addressed by the configured product.

In the DSPL context, it is important to specify a mechanism which can reconfigure the product to reflect the current requirements of the client. For instance, in our running example, despite the hotel having alarm, the feature *Bliking lights* was not originally selected to be in the Smart Hotel configuration. However, in a changing scenario, the client would like to include it in his product.

To achieve such change, it is necessary to change the property that indicates that the feature is present or not in the system in the ontology of the product at runtime. The update of this property can be realized by the SPARQL query on Figure 4.

```
PREFIX spl:<http://nees.com.br/ontologies/SPL.owl#>
DELETE {
?x spl:current_state "eliminated".}
INSERT {
?x spl:current_state "selected".}
WHERE{?x spl:name "BlikingLights"}
```

Figure 4. Selecting An Optional Feature in a Product.

As can be seen in the query, after setting the status of the *Bliking Lights* feature as ”Selected”, such feature is included in the product.

3) Changing an Alternative Feature: Usually, it is necessary to select alternative ways to realize a product requirement. The alternative type of features specifies a design space of variations on which a product can use. A product configuration requires the selection of one of the variants on such kind of features, but it is possible that a client would be not satisfied with the variant selected and wants to reconfigure a product with another variant.

For instance, in our running example, the selected variant of alarm is the silent type. However, one could require to change from the *Silent Alarm* feature to the *Siren* one.

Figure 5 specifies a SPARQL query which makes such variant change. This query is similar to the one presented in Subsection III-B2, however, it is not only a selection of a new feature, but rather the substitution of one feature by another. Thus, there is the need to remove the existing feature and then, add the new feature.

```
PREFIX spl:<http://nees.com.br/ontologies/SPL.owl#>
DELETE{
?x pSPL:current_state "selected".
?y pSPL:current_state "eliminated"
}
INSERT{
?x pSPL:current_state "eliminated".
?y pSPL:current_state "selected"
}
WHERE{
?y spl:name "Siren".
?x spl:name "SilentAlarm"
}
```

Figure 5. Changing an Alternative Feature.

As defined in the query, the *Silent Alarm* would be eliminated from the hotel and the *Siren* would be enabled in the product. After executing this query, the selected type of *Alarm* is the *Siren* feature.

IV. Related Works

Using ontologies in the development of SPLs has been addressed by several studies in the literature. In fact, it was found some studies with the particular aim of using ontology for representing feature models [3][4]. Furthermore, we have

 ISBN: 978-1-61208-367-4

found only one study which makes use of ontology-based feature modeling in the design of DSPLs [10].

Wang et al. [3] presents a technique to design ontology-based feature models, by representing feature models as OWL classes and properties. Moreover, they use OWL reasoning engines to check for inconsistences of feature configurations automatically. However, since the features in such modeling are represented as OWL classes and properties, every change in the feature model requires a structural modification in the ontology.

OntoSPL ontology presents an alternative way for modeling ontology-based feature models. It proposes a predefined structure of classes and properties and suggests the creation of features model as OWL instances/individuals of such structure. This alternative way for feature modeling is more suitable than the one of [3] for dynamic reconfigurations of features.

In order to change a feature model using Wang's ontology [3], for instance, adding a new feature, it would be necessary to change the structure of the ontology and then it would be also necessary to generate the ontology mapping code again. Thus, applying such changes in an application would require to stop the execution of the system. On the other hand, using the OntoSPL, changes are performed at the instances level. This characteristic allows to change instances at runtime, i.e., it is not necessary to generate the ontology mapping code again and hence, it would not be necessary to stop the application.

The work by Zaid et al. [4] also presents an ontology to represent feature models based on OWL instances which is similar to our ontology. However, it is focused on the automatic consistency verification of feature models and it was not conceived to support dynamic reconfiguration of features. Thus, it does not consider important issues regarding changes at runtime, for example, properties related to the status of the feature.

Regarding the use of ontology in the development of dynamic Software Product Lines, Kaviani et al. [10] use ontology to annotate feature models covering non-functional requirements modeling in the context of ubiquitous environments. However, it also represents feature models as proposed in [3], thus the same limitations regarding the impact of changes in the feature model are also applicable to it. Moreover, it is noteworthy that the dynamic reconfiguration effort using the OntoSPL ontology is lower, since it is only necessary to change product configurations through SPARQL queries without needing to generate code to manipulate a product.

V. Conclusion and Future works

In this paper, we presented the OntoSPL ontology, which is used to specify feature models in a formal way with the special aim to support automatic reconfigurations of products in the context of Dynamic Software Product Line. To illustrate how to reconfigure DSPL products using such ontology, we also specified three SPARQL queries that were applied to an existent running example in the literature.

This study can be considered as a first step towards selecting a suitable way for formalizing feature models to be used in the context of DSPLs. Future works should include the conduction of a controlled experiment in different contexts to evaluate the effectiveness of OntoSPL in comparison with other ontologies regarding its capabilities (e.g., time to realize some change, flexibility and so on) for performing reconfiguration at runtime. Moreover, we intend to incorporate some consistency checking mechanism in OntoSPL to validate product reconfigurations. We also intend to define a DSPL process based on the OntoSPL.

Acknowledgments

This work has been supported by the Brazilian institutions: Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) and Coordenacão de Aperfeicoamento de Pessoal de Nível Superior (CAPES).

References

[1] K. Pohl, G. Bockle, and F. Van Der Linden, Software product line engineering. Springer, 2005, vol. 10.

[2] M. Hinchey, S. Park, and K. Schmid, "Building dynamic software product lines," Computer, vol. 45, no. 10, 2012, pp. 22–26.

[3] H. H. Wang, Y. F. Li, J. Sun, H. Zhang, and J. Pan, "Verifying feature models using owl," Web Semantics: Science, Services and Agents on the World Wide Web, vol. 5, no. 2, 2007, pp. 117–129.

[4] L. A. Zaid, F. Kleinermann, and O. De Troyer, "Applying semantic web technology to feature modeling," in Proceedings of the 2009 ACM symposium on Applied Computing. ACM, 2009, pp. 1252–1256.

[5] N. Guarino, Formal Ontology in Information Systems: Proceedings of the 1st International Conference June 6-8, 1998, Trento, Italy, 1st ed. Amsterdam, The Netherlands, The Netherlands: IOS Press, 1998.

[6] K. Czarnecki, C. H. Peter Kim, and K. T. Kalleberg, "Feature models are views on ontologies," in Proceedings of the 10th International on Software Product Line Conference, ser. SPLC '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 41–51.

[7] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study, 1990.

[8] C. Cetina, P. Giner, J. Fons, and V. Pelechano, "Prototyping dynamic software product lines to evaluate run-time reconfigurations," Science of Computer Programming, vol. 78, no. 12, 2013, pp. 2399 – 2413, special Section on International Software Product Line Conference 2010 and Fundamentals of Software Engineering (selected papers of {FSEN} 2011).

[9] W3C, "Sparql 1.1 query language," Available in http://www.w3.org/TR/2012/PR-sparql11-query-20121108/, retrieved: October, 2014.

[10] N. Kaviani, B. Mohabbati, D. Gasevic, and M. Finke, "Semantic annotations of feature models for dynamic product configuration in ubiquitous environments," in Proceedings of the 4th International Workshop on Semantic Web Enabled Software Engineering, in collaboration with International Semantic Web Conference (ISWC), Karlsruhe, Germany, 2008.

 ISBN: 978-1-61208-367-4

A Formal Model of Use-Cases and Its Application in Generating A Hierarchical Class-Structure

Sukhamay Kundu

Department of Computer Science
Louisiana State University, Baton Rouge, LA 70803
Email: kundu@csc.lsu.edu

Arnab Ganguly

Department of Computer Science
Louisiana State University, Baton Rouge, LA 70803
Email: agangu4@lsu.edu

***Abstract*—Creating an object-oriented design from user requirements, given as a set of use-cases, means deriving a detailed class structure that can support an implementation of those requirements. We introduce here the Augmented Finite-State (AFS) model for a set of use-cases. An AFS model of a single use-case U incorporates the inputs, outputs, and operations for each interaction in U, including the "internal" dataflows among those interactions. The AFS model for a set of use-cases $\mathcal{U}$ combines the AFS models of individual use-cases $U_j \in \mathcal{U}$ to account for common interactions among U_j's and the control-flows among U_j's. After we decompose the combined model into a unique set of disjoint Maximal Linear Segments (MLSs), we derive one class from each MLS and finally create the class-hierarchy based on the next-relationship among the MLSs. One advantage of our approach over those based on the concept-analysis is that the AFS model gives a simple controller for the call-sequences of the class-methods corresponding to each U_j.**

Keywords–Augmented finite-state model; class hierarchy; object-oriented design; refactoring; use-case model.

I. INTRODUCTION

Software design is an essential part of any software development effort. For an object-oriented software, the design consists of the classes (their attributes, methods, and method-parameters) and the relationships among those classes, which includes the class-hierarchy and other associations. The design gives a global view of the functionalities and structure of the software, and plays a critical role in understanding, implementation, and analysis of the software.

Semi-automated generation of UML-models and class-diagrams from natural language description of requirements are discussed in [1][2]. Automated generation of UML-models are discussed in [3][4]. These works are based on Natural Language Processing, and rely on use-cases defined using a semi-formal syntaxes and semantics. Cockburn [5] argued that there is no formal syntax or semantics for writing use-cases. Roussev [6] uses an informal notion of "balance" of objects involved in a use-case, and assumes the use-cases are given in terms of pre-conditions, post-conditions and invariants on the objects involved in the use-cases. The identification of objects is a key missing step in [6]; also, the notion of "balance" of objects has a basic flaw because "information" do not behave like the physical quantities force, energy, and mass, and we don't have a principle like the "conservation of energy" for "information".

Modeling means choosing a proper abstraction and a suitable representation of it to facilitate its use. Finite-state models and interaction-diagrams are often used in explaining a class-structure design [7]. We use a reverse approach: we first create an Augmented Finite-State (AFS) model of the use-cases (interactions) that describe the requirements and then we build the classes and their relationships from this model. This gives a more systematic and precise (semi-formal) technique compared to the other methods in the literature. A class design involves identification and grouping of operations and their supporting variables (inputs and outputs of the operations, and other intermediate stored data to avoid recomputation) in a way that minimizes the information overload. The AFS model facilitates both of these steps by capturing the essential operational details of the system's functional requirement. The identification of operational details for each use-case plays a key role in our approach. The method presented here can be regarded as a refinement of that in Kundu [8].

Many models are used in software engineering as effective tools. For example, *Finite-state* machines are used by Chow [9] for automated software testing. Our AFS model has some resemblance to *X-machines* [10]. In *X-machines*, a transition between two states is labeled by an operation whereas in AFS the labels are constraints; the dataflow items in AFS model correspond to the concept of "memory" in *X-machines*.

In Section II, we give the detailed formal definition of a use-case, and Section III defines the AFS model of a set of use-cases Sections IV and V explain our AFS-based approach for generating a class-hierarchy using a simplified set of use-cases for a bank's ATM machine. Section VI provides a brief conclusion.

II. A FORMAL DEFINITION OF USE-CASE

Jacobson [11] defines a use-case as a sequence (chain) of interactions $U = \langle t_1, t_2, \cdots, t_n \rangle$, which provides the user a useful service, i.e., corresponds to a complete high-level functional requirement. If U_1 and U_2 are two use-cases, then clearly U_1U_2 is also a use-case. Henceforth, a use-case U will mean an *elementary* use-case, which cannot be decomposed into a sequence of two or more disjoint smaller use-cases.

A. Interaction t_j

Formally, an interaction $t_j = (in_j, op_j, out_j)$ is a triplet, where $in_j = in(t_j)$ is a set of input data-items, $op_j = op(t_j)$ is an operation, and $out_j = out(t_j)$ is a set of output data-items.

 ISBN: 978-1-61208-367-4

The inputs in_j consist of two disjoint parts: user-inputs in^u_j and other "external" (with respect to U) inputs in^e_j. The term "interaction" means $in^u_j \neq \emptyset$; however, one possible exception to this is that the last interaction t_n in U may have $in^u_n = \emptyset$ (cf. Theorem 1). An operation op_j may have multiple parts and a user may provide different parts of in^u_j at different stages of op_j, with in_j and out_j interleaved. The inputs to op_j may also include zero or more additional "internal" data-items d_i that are generated by previous op_i's, $i < j$ and $t_i \in U$. We refer to such d_i's as *dataflows*. The outputs out_j also consists of two parts: user-outputs out^u_j and external-outputs out^e_j; because parts of out^u_j maybe saved for use in other use-cases and hence included in out^e_j, we may have $out^u_j \cap out^e_j \neq \emptyset$. Clearly, $in_j = in^u_j \cup in^e_j$ and $out_j = out^u_j \cup out^e_j$. Henceforth, we use t_j and op_j interchangeably when no confusion is likely. Figure 1 shows the structure of an interaction t_j. We can regard op_j as a function of in_j and zero or more d_i, $i < j$, i.e., $in(op_j) = \bigcup_{i \leq j} in_i$. In contrast, $in(t_j) = in_j$ and thus t_j is not exactly the same as op_j. The dataflow d_j is not determined by op_j but by $t_k, k > j$ in U; also, d_j need not be a subset of out_j.

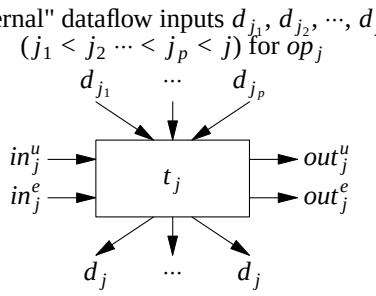


Figure 1. Structure of an interaction t_j.

B. Use-Case U

We formally define a use-case as a sequence of interactions $U = \langle t_1, t_2, \cdots, t_n \rangle$ with the properties (1)-(4) below.

(1) $in^u_i \cap in^u_j = \emptyset$ for $i \neq j$. A user should not be required to provide the same input more than once in a use-case. If $op_j, j > i$, requires parts of the user-input in^u_i, then op_i may include those parts of in^u_i in the internal dataflow item d_i generated by op_i. However, $out^u_i \cap out^u_j$ may be non-empty because parts of out^u_i might be repeated in out^u_j (e.g., a "confirm operation" prompt to the user for a critical operation like deleting a file). We assume each in_j is as small as possible, i.e., no unnecessary "early" inputs and all of in_j is used in op_j. Likewise, we assume each out_j is as large as possible, i.e., no "late" outputs. For efficient input/output operations involving files and databases, one may want to maximize each chunk of information exchange but for modeling purpose these "early" and "late" viewpoints are more logical.

(2) $in^e_i \cap in^e_j = \emptyset = out^e_i \cap out^e_j$ for $i \neq j$. As before, if $op_j, j > i$, requires parts of in^e_i, then op_i may include those parts of in^e_i in d_i. This is desirable if accessing the external input data-items are computationally expensive. We may also include parts of out_i in d_i to avoid recomputing them in $op_j, j > i$. (A non-empty $out^e_i \cap out^e_j$ would mean parts of out^e_i is overwritten by op_j, $j > i$, based on additional information available at op_j.) Because op_j does not use any in_k or d_k, $k > j$, there is no *cyclic dependency* among op_j's.

(3) $d_i \cap d_j = \emptyset$ for $i \neq j$. Any part of d_i can be made available to each $op_j, j > i$, as needed.

(4) Each (t_j, t_{j+1})-pair has an associated *transition-condition* $c_{j,j+1}$ which needs to be satisfied after op_j is completed in order for op_{j+1} to start; $c_{j,j+1} = true$ means the condition is trivially satisfied. The condition $c_{j,j+1}$ does not depend on in_k, $k > j$, and may depend only on parts of $in(op_j)$. All non-trivial conditions $c_{i,i+1}, i \leq j$, contribute directly or indirectly to the pre-condition for op_{j+1}. We assume for now that $c_{j,j+1}$ is evaluated by op_j.

We write $I^u_U = \bigcup in^u_j$ and $I^e_U = \bigcup in^e_j$, where the unions are taken over all j, and finally $I_U = I^u_U \cup I^e_U$. Similarly, we write $O^u_U = \bigcup out^u_j$, $O^e_U = \bigcup out^e_j$, $O_U = O^u_U \cup O^e_U$, $D_U = \bigcup d_j$, and $C_U = \{c_{j,j+1} : 1 \leq j < n\}$. The entities in I_U, O_U, and D_U are the names of data-items and not any specific values for them. (An instance of a use-case U, with concrete values for the data-items in I_U and hence concrete values for the data-items in $D_U \cup O_U$, is called a scenario.)

If the use-case U' is used after the use-case U and we need to use parts of in^u_j at $t_j \in U$ as parts of $in^u_{j'}$ at $t_{j'} \in U'$, then we can include those parts of in^u_j into out^e_j and those parts of out^e_j can now become a part of $in^e_{j'}$. This avoids having to provide the common parts of in^u_j and $in^u_{j'}$ more than once.

A proper choice of the individual interactions t_j in modeling a use-case U is a non-trivial task. A simpler or smaller t_j can help to reduce errors in determining $in(op_j), out(op_j)$, and $c_{j,j+1}$, but it can also introduce unnecessary details in the design of a class-structure for U. A complex or larger t_j can, on the other hand, prevent sharing interactions between different use-cases. These issues are described next.

C. Merging Interactions

If we merge $t_j, t_{j+1} \in U$ into a single interaction $t_{j,j+1}$, then we have $in^u_{j,j+1} = in^u_j \cup in^u_{j+1}$ and similarly for $in^e_{j,j+1}$, $out^u_{j,j+1}$, and $out^e_{j,j+1}$. In general, $d_{j,j+1} \subseteq d_j \cup d_{j+1}$, with $t_{j,j+1}$ hiding dataflows from t_j to t_{j+1} (making them internal to $op_{j,j+1}$); in the extreme case, we may have $d_{j,j+1} = d_{j+1}$. Also, $op_{j,j+1} = op_j \circ op_{j+1}$, the composition (roughly speaking in view of $c_{j,j+1}$) of op_j and op_{j+1}, in that order. The new interaction sequence $\langle t_1, t_2, \cdots, t_{j-1}, t_{j,j+1}, t_{j+2}, \cdots, t_n \rangle$ has less information about $op_{j,j+1}$ (equivalently, $t_{j,j+1}$) because we know less about which parts of $in_{j,j+1}$ are used by which parts of $op_{j,j+1}$ to produce which parts of $out_{j,j+1}$ and $d_{j,j+1}$. There is, however, no change in I^u_U, I^e_U, O^u_U, and O^e_U.

Except for the loss of some information as noted above, it is safe to merge t_j and t_{j+1} when $c_{j,j+1} = true$. In this case, each use-case U' containing t_j will also contain t_{j+1} and thus we can replace t_j and t_{j+1} by $t_{j,j+1}$ in each U'.

The merging of t_j and t_{j+1} has no impact on the condition from t_{j-1} to $t_{j,j+1}$, i.e., $c_{j-1,(j,j+1)} = c_{j-1,j}$, which will be evaluated by op_{j-1}. However, determining the condition $c_{(j,j+1),j+2}$ from $t_{j,j+1}$ to t_{j+2} might pose a problem as shown below. Consider the situation on the left-side in Figure 2. The part $\bar{c}_{j,j+1}$: "$x + z \neq 0$" in $c_{(j,j+1),j+2} = \bar{c}_{j,j+1} \wedge c_{j+1,j+2}$ shown on the rightside in Figure 2 is the result of "pushing down" the condition $c_{j,j+1}$: "$x \neq 0$" through op_{j+1}, which gives $x_{new} = x_{old} - z$, i.e., $x_{old} = x_{new} + z$, and thus $c_{j,j+1}$: "$x \neq 0$" = "$x_{old} \neq 0$" becomes $\bar{c}_{j,j+1}$: "$x_{new} + z \neq 0$" = "$x + z \neq 0$". But a difficulty arises if we replace "$x = x - z$" in op_{j+1} by "$x = x^2 - z$" because we cannot express now x_{old} in

terms of x_{new}. We can, however, add the computation "$x_{old} = x$" at the end of op_j and call it op'_j, let $op_{j,j+1} = op'_j \circ op_{j+1}$ and $c_{(j,j+1),j+2}$: $(x_{old} \neq 0) \wedge (y > x)$. But such tricks do not always suffice as would be the case if "$x = x - z$" in op_{j+1} is replaced by "$x = x - z/x$"; op_{j+1} now needs $x \neq 0$ and thus we cannot afford to do the test "$x_{old} \neq 0$" after $op_{j,j+1}$.

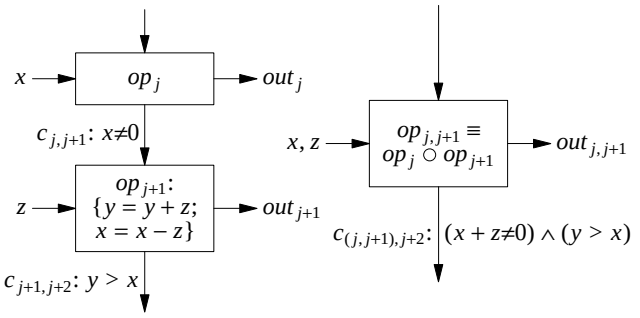


Figure 2. Illustration of a simple case of $c_{(j,j+1),j+2}$.

Even if we could define a suitable $c_{(j,j+1),j+2}$, the creation of $t_{j,j+1}$ would prevent sharing just one of t_j and t_{j+1} with another use-case U', and this can be a good reason not to create $t_{j,j+1}$. On the other hand, if we have a use-case $U' = \langle \cdots, t_j, t_{j+1}, t'_{j+2}, \cdots \rangle$ containing both t_j and t_{j+1} but a different t'_{j+2}, with $c'_{j+1,j+2}$ = "$y \leq x$" = $\neg c_{j+1,j+2}$ for the pair (t_{j+1}, t'_{j+2}), then after we create $t_{j,j+1}$ we will have $c'_{(j,j+1),j+2} = (x + z \neq 0) \wedge (y \leq x)$ in U'. Obviously, we cannot merge t_j and t_k, $k > j + 1$, and keep U acyclic.

Theorem 1. For a use-case $U = \langle t_1, t_2, \cdots, t_n \rangle$, there is no loss of generality to assume that each $in_j \neq \emptyset$ for $j < n$ if some use-cases U' share $t_i \in U$ exactly upto t_j.

Proof. If $in_j = \emptyset$ and $j < n$, then $c_{j,j+1}$ depends only on $\bigcup_{i<j} in_i$ and thus we can create the shortened use-case $\underline{U}$ by merging t_j into t_{j+1} as follows. We let $c_{j-1,j+1} = c_{j-1,j} \wedge lift(c_{j,j+1})$, where $lift(c_{j,j+1})$ is the result of "lifting up" $c_{j,j+1}$ through op_j, and replace op_{j+1} by $\underline{op}_{j+1} = op_j \circ op_{j+1}$. Note that $c_{j-1,j+1}$ can be evaluated by op_{j-1}. If there is another use-case $U' = \langle \cdots, t_{j-1}, t_j, t'_{j+1}, \cdots \rangle$, which is identical to U upto t_j, then we can likewise create the shortened use-case $\underline{U'}$ by merging t_j into t'_{j+1}, with op'_{j+1} replaced by $\underline{op}'_{j+1} = op_j \circ op'_{j+1}$ and letting $c'_{j-1,j+1} = c_{j-1,j} \wedge lift(c'_{j,j+1})$, where $c'_{j,j+1}$ is the condition for the pair (t_j, t'_{j+1}). Note that $c_{j-1,j+1} \wedge c'_{j-1,j+1} = c_{j-1,j} \wedge lift(c_{j,j+1}) \wedge lift(c'_{j,j+1}) = c_{j-1,j} \wedge lift(c_{j,j+1} \wedge c'_{j,j+1}) = c_{j-1,j} \wedge lift(false) = c_{j-1,j} \wedge false = false$, as desired. The shortened use-cases $\underline{U}$ and $\underline{U'}$ now share only upto t_{j-1}. □

Two remarks are due here. First, merging t_j with t_{j+1} to avoid $in_j \neq \emptyset$ does not cost us in terms of its effect on the class design. The methods for $\underline{op}_{j+1}$ and $\underline{op}'_{j+1}$ in the classes for $\underline{U}$ and $\underline{U'}$ will now have some commonalities because op_j is a part of both $\underline{op}_{j+1}$ and $\underline{op}'_{j+1}$. However, we can refactor the common part, if needed, to a parent class. Second, we do not merge t_j into t_{j-1} in the proof of Theorem 1 because if there is an use-case U'' that is identical to U only upto t_{j-1} then the merging would create an overloaded $t_{j-1,j}$ in terms of outputs and the operation $op_{j-1,j} = op_{j-1} \circ op_j$, and this can cause problems with the condition for $(t_{j-1,j}, t''_j)$-pair for U''. If there is no U'', we could form $t_{j-1,j}$ to eliminate t_j.

D. Decomposing an Interaction

If we can decompose an op_j into a chain of suboperations $\langle op_{j.1}, op_{j.2}, \cdots, op_{j.m} \rangle$, $m \geq 2$, then should we replace t_j in U by the chain of interactions $\langle t_{j.1}, t_{j.2}, \cdots, t_{j.m} \rangle$, where $op_{j.p}$ corresponds to $t_{j.p}$? If we did, then we will have $c_{j-1,(j.1)} = c_{j-1,j}$, $c_{(j.m),j+1} = c_{j,j+1}$, and $c_{j.p,j.(p+1)} = true, 1 \leq p < m$. This implies that it is safe to merge $t_{j.p}$'s and hence the decomposition is unnecessary. Note that because $c_{j.p,j.(p+1)} = true$, there is no use-case U' that includes $t_{j.p}$ but not $t_{j.(p+1)}$.

E. Deleting an Interaction

In general, the deletion of a $t_j \in U$ may not give a valid use-case $\langle t_1, t_2, \cdots, t_{j-1}, t_{j+1}, t_{j+2}, \cdots, t_n \rangle$. For example, if t_k, $k > j$, requires d_j generated by op_j then removal of t_j makes op_k inapplicable; hence t_k needs to be removed. This may, in turn, require other t_m, $m > k$, to be removed and so on. On the other hand, if $k > j$ is the smallest index such that t_k requires d_j then $\langle t_1, t_2, \cdots, t_{j-1}, t_{j+1}, t_{j+2}, \cdots, t_{k-1} \rangle$ may not be a valid use-case because the output of t_{k-1} may involve a prompt to the user to provide an input ($in^u_k \neq \emptyset$). The same argument shows that deletion of t_{k-1} may create a problem, and so on. A similar argument shows that an initial part of a use-case may not be a valid use-case. Likewise, a tail part $\langle t_{k+1}, t_{k+2}, \cdots, t_n \rangle$ of a use-case may not be a valid use-case because the output of t_k may involve a prompt to the user to provide an input and without that prompt t_{k+1} becomes meaningless.

III. Augmented Finite State (AFS) Model

The AFS model of a set of use-cases $\mathcal{U}$, denoted by $AFS(\mathcal{U})$, combines the notions of finite state machines, flowcharts, and Dataflow Diagrams (DFDs, which can be regarded as high-level dataflow-abstractions of flowcharts). As a finite-state machine, each state s_j in $AFS(\mathcal{U})$ corresponds to an interaction t_j in a use-case in $\mathcal{U}$. Each transition (s_j, s_k) corresponds to the next-interaction t_k of t_j in a use-case in $\mathcal{U}$ that contains t_j, and associated with the transition (s_j, s_k) we have the corresponding condition $c_{j,k}$. Clearly, $c_{j,k}$ is independent of the use-case in which t_k is the next interaction after t_j, and it can be likened to a branching-condition in a flowchart. We also have the dataflows d_i between interactions or states. As before, we consider a state $s_j = t_j$ to be synonymous with the operation op_j associated with t_j. If $|\mathcal{U}| = 1$, then $AFS(\mathcal{U})$ takes the form of a single chain. The condition $c_{j,k}$ associated with transition (s_j, s_k) must be satisfied for the transition to take place. For two transitions (s_j, s_k) and (s_j, s'_k), $s_k \neq s'_k$, the conditions $c_{j,k}$ and $c_{j,k'}$ must be disjoint, i.e., $c_{j,k} \wedge c_{j,k'} = false$. Unlike a flowchart, an $AFS(\mathcal{U})$ by definition does not have a cycle and this prevents cyclic data-dependencies. The usual use-dependencies among data-items in assignments and other computations in a flowchart are replaced in $AFS(\mathcal{U})$ by the higher-level abstractions inputs, outputs, and dataflows associated with a state.

We formally define $AFS(\mathcal{U})$ = (S, s_0, S_{final}, C, D, I^u, I^e, O^u, O^e, $\tau, \delta, \phi^u, \psi^u, \phi^e, \psi^e$), where

1) $S \neq \emptyset$ is a set of states and $s_0 \in S$ is the start-state; each state s_j is reachable from s_0 by a sequence of transitions and has an associated operation op_j.

2) $S_{final} \subseteq S$ is the subset of final (terminal) states, from which there are no transitions.
3) C = {$c_{i,j}$: conditions associated with transitions (s_i, s_j)} = $\bigcup C_U$, union over the use-cases $U \in \mathcal{U}$.
4) D = {d_i: the internal dataflow item generated by op_i at s_i} = $\bigcup D_U$, union over the use-cases $U \in \mathcal{U}$.
5) $I^u = \bigcup I^u_U$, $I^e = \bigcup I^e_U$, $O^u = \bigcup O^u_U$ and $O^e = \bigcup O^e_U$, each union over the use-cases $U \in \mathcal{U}$.
6) $\tau : S \times C \to S$ is the transition function.
7) $\delta : D \to S \times \mathcal{P}^+(S)$ is the function representing the dataflows, where $\mathcal{P}^+(S)$ = the set of non-empty subsets of S, $\delta(d_j) = (s_j, S_j) = (\delta_1(d_j), \delta_2(d_j))$ means S_j = {s_k: s_k uses d_j generated at s_j}, and $d_i \neq d_j$ means $s_i = \delta_1(d_i) \neq \delta_1(d_j) = s_j$.
8) $\phi^u : I^u \to \mathcal{P}^+(S)$ is the user-input function; $\phi^u(u')$ = {s_k: s_k requires user-input u'}.
9) $\phi^e : I^e \to \mathcal{P}^+(S)$ is the external-input function; $\phi^e(e')$ = {s_k: s_k requires external-input u'}.
10) $\psi^u : O^u \to \mathcal{P}^+(S)$ is the user-output function.
11) $\psi^e : O^e \to \mathcal{P}^+(S)$ is the external-output function.
12) For each $s_i \notin S_{final}$, the conditions $c_{i,j}$ are mutually disjoint (i.e., $c_{i,j} \wedge c_{i,k} = false$ for $j \neq k$). We may also assume that $\bigvee c_{i,j} = true$ because otherwise we can add a "graceful" exit-transition to a new final state s'_i with the transition-condition $c_{i,i'} = \neg \bigvee c_{i,j}$.
13) For any $s_i, s_j \in \phi^u(u')$ for an user-input u', neither of s_i and s_j is reachable from the other. Similarly for ϕ^e and ψ^e. (But this is not required for ψ^u.)

In what follows, we consider only the special case where $AFS(\mathcal{U})$ has a tree-structure. Recall that not all paths in a flowchart, even in absence of cycles, may represent a valid execution-path; likewise, if the transitions in $AFS(\mathcal{U})$ form a general acyclic digraph, then we may have paths from the start-state s_0 to a final-state that do not represent a valid use-case and this can severely complicate the derivation of a suitable class-structure from $AFS(\mathcal{U})$. In the case of a tree-structured $AFS(\mathcal{U})$, each path from the start-state s_0 to a final-state represents a valid use-case in $AFS(\mathcal{U})$.

$AFS(\mathcal{U})$ helps us to see the relationships among the use-cases $\mathcal{U}$ in terms of their shared interactions. In particular, it helps us to identify inconsistencies inconsistent order of operations, invalid dataflow dependencies, and missing inputs/outputs for the use-cases $\mathcal{U}$. One must, indeed, resolve all inconsistencies before attempting to create a class-structure from $\mathcal{U}$ for the desired software.

IV. METHODOLOGY

Given the AFS-model of a single use-case U, we use the Class-Creation-Rules below to obtain a class that supports an implementation of U. These rules can be used also, more generally, for any linear chain of interactions. Initially, the class-methods have no parameters and this has the advantage of a simple control mechanism for executing the methods in a class (see Section V-C). We may later use refactoring to introduce new methods (possibly, with parameters) for common or similar parts of the original class-methods, and replace the common parts in the original methods by calls to the new refactored methods with suitable parameter-values.

Class-Creation-Rules for a single use-case U:
1. The variables are the internal dataflows, which may include parts of $I_U \cup O_U$.
2. The class-methods are the operations $op_j \in U$ or parts of them.

If $|\mathcal{U}| > 1$, we can first create one class for each use-case in $\mathcal{U}$ and then refactor common class-variables and methods to create the final class-structure. A better method is to build the class-structure directly from the combined model $AFS(\mathcal{U})$, whose tree-structure directly leads to a tree-structured class-hierarchy, with one class for each maximal linear segment (MLS) of the tree. A linear segment in $AFS(\mathcal{U})$ is a path π in $AFS(\mathcal{U})$, where each state in π other than those at the start and end of π has a single child (next) node.

We remark that the notation $t_j = (in_j, op_j, out_j)$ implies that if $t_j \in U$ is shared and equals $t'_j = (in'_j, op'_j, out'_j) \in U'$, then $in_j = in'_j$, $op_j = op'_j$, and $out_j = out'_j$. However, the dataflow d_j from op_j in U may differ from the dataflow d'_j from op'_j in U'. For a tree-structured $AFS(\mathcal{U})$, with $U, U' \in \mathcal{U}$, $t_j = t'_j$ implies $op_j = op'_j$ can compute $d_j \cup d'_j$, and thus we can replace both d_j in U and d'_j in U' by $d_j \cup d'_j$. Viewed another way, this simply points out that while the classes for U and U' obtained by the Class-Creation-Rules may contain different class variables due to $d_j \neq d'_j$, when we merge those classes to create a class-hierarchy the class containing $op_j = op'_j$ can include the variables for both d_j and d'_j.

We illustrate below our method by deriving a class-structure for a bank's ATM-system with three high-level functional requirements or use-cases $\mathcal{U}$ = $\{U_1, U_2, U_3\}$, where U_1 = successful withdrawal, U_2 = failed withdrawal due to insufficient funds, and U_3 = balance enquiry. We first derive a class for each U_j using the Create-Class-Rules and then show that the class-structure obtained by refactoring these classes can be obtained directly from $\mathcal{U}$.

A. Informal Description of ATM

A user swipes a debit card in the ATM's card-slot. The ATM reads the debit card and prompts the user to enter the PIN. We assume for simplicity that no invalid ATM card or PIN is used, and there is no cash dispenser malfunction. The ATM validates the PIN and asks the user to choose one of two displayed options "withdrawal" and "balance-enquiry". If the user selects withdrawal-option, the ATM calculates and displays the maximum allowable withdrawal amount based on the available ATM cash and debit-card-account-information. Then, the ATM asks the user to enter the withdrawal amount and it reads that amount. Then, either the ATM displays the updates to debit-card-account-info and dispenses the desired cash, or it displays a transaction-fail-message when withdrawal amount is too large. If the user selects balance-enquiry option, the ATM displays the debit-card-account-balance. In each case, the ATM writes a transaction-log for future audit analysis as part of session-closing operation.

B. Formal Description of Use-Case U_1

We show below the decomposition of U_1 into four interactions t_1-t_4 and also show each op_j in detail, including its lower level operations. This simplifies the identification of inputs in_j, outputs out_j, dataflow items d_j, and the transition conditions $c_{j,j+1}$ for U_1, which are shown in Tables I and

II. The data-item ATMidAndOtherInfo in in_3^e includes ATM-cash-balance among others. The Transaction-Log-Information (TLI) has many parts, including dateTime of transaction, transaction amount, update of Bank-Debit-card-Account-Detail-Info, etc. Different out_j^e includes different parts of TLI.

U_1: Successful withdrawal

t_1: User swipes the debit card in the ATM's card slot and enters PIN when requested. [**op_1:** ATM reads the debit card number, then reads Bank-Debit-Card-PIN Information based on the debit card number, displays "Enter PIN" message, reads the PIN entered and validates it with Bank-Debit-Card-PIN information, reads Bank-Debit-Card-Account-Detail Information for the debit card number, and writes part-1 (debit card number, PIN, and transaction dateTime) of TLI]

t_2: User sees the displayed transaction options and selects the "withdrawal" option. [**op_2:** ATM displays the transaction-options "balance-enquiry" and "withdrawal", requests the user to select an option, reads the selected option (= "withdrawal" for U_1), and writes part-2 (transactionOption = withdrawal) of TLI.]

t_3: User sees the displayed max allowed withdrawal amount and enters the desired withdrawal amount. [**op_3:** ATM calculates and displays the max allowed withdrawal amount based on the ATM-cash-balance (which is part of the external input ATMidAndOtherInfo) and the debit-card-account-detail information (such as the single-transaction-limit, the daily-withdrawal-limit for the card, today's-current-total-withdrawal, and the debit-card-account-balance). Then, ATM requests user to enter the desired-withdrawal-amount, reads it, and writes part-3 (max allowed withdrawal amount) of TLI.]

t_4: User takes the dispensed cash. [**op_4:** ATM updates ATM-cash-balance part of ATMidAndOtherInfo and the Bank-Debit-Card-Account-Detail information (such as today's-total-withdrawal and debit-card-account-balance), displays the updated debit-card-account-balance, displays "collect-cash" message (which includes the amount withdrawn), dispenses cash for the withdrawal amount, and writes part-4 (updates of ATM-cash-balance, this withdrawal amount, debit-card-account-balance) of TLI.]

TABLE I. Inputs in_j and outputs out_j for U_1

in_1	u: Debit-card number, PIN number e: Bank-Debit-Card-PIN Info, Bank-Debit-Card-Account-Detail Info	out_1	u: "Enter PIN" message e: Part-1 of TLI
in_2	u: Selected Transaction Option	out_2	u: Transaction-options display, "Select Option" message e: Part-2 of TLI
in_3	u: Desired Withdrawal Amount ATM-id-And-Other Info	out_3	u: Max Allowable Withdrawal Amount display, "Enter Desired Withdrawal Amount" message e: Part-3 of TLI
in_4	u: Cash Collected	out_4	u: New BankAccountDetailInfo, "Collect Cash" message e: New BankAccountDetailInfo, New ATMidAndOtherInfo, Part-4 of TLI

TABLE II. Conditions $c_{j,j+1}$ and data-items d_j for U_1

$c_{1,2}$	true (no invalid card or PIN)	d_1	Bank-Debit-Card-Account-Detail Info
$c_{2,3}$	"Withdrawal" = Selected Transaction Option	d_2	Selected Transaction Option
$c_{3,4}$	Max Allowable Withdrawal Amount ≥ Desired Withdrawal Amount	d_3	Max Allowable Withdrawal Amount, Withdrawal Amount Desired

In general, the conjunction of $c_{j,j+1}$'s for the interaction-sequence of a use-case U does not give a pre-condition for U because each $c_{j,j+1}$ is stated in terms of values of data-items "after" the operation op_j. For U_1, $c_{2,3} \wedge c_{3,4}$ does give its pre-condition. We consider d_2 to be an "implicit" dataflow from op_2 to op_3 because execution of op_3 requires $c_{2,3}$ to be true. Similarly, we consider d_3 to be an implicit dataflow from op_3 to op_4. (The controller to drive the execution of the methods in the class for U_1 will use d_2 and d_3; see Section V-C.) See Figure 3, which shows the finite-state machine model and the dataflow model for U_1. There is no dataflow from t_1 to t_2.

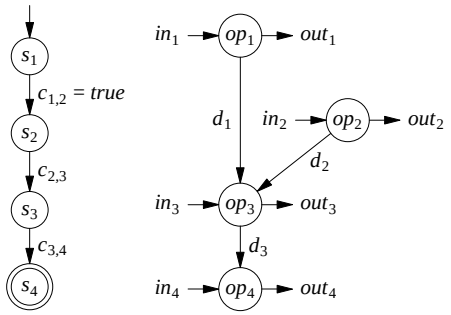
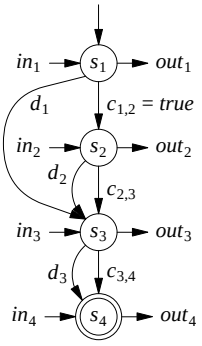


Figure 3. FSM (left) and DFD (right) for the use-case U_1



SuccessfulWithdrawal	
ATMidAndOtherInfo	
bankDebitCardPINinfo	
bankDebitCardAccountDetailInfo	// d_1
selectedTransactionOption	// d_2
maxAllowableWithdrawalAmount	// d_3
desiredWithdrawalAmount	// d_3
readDebitCardPINinfo()	// op_1
displayEnterPINmssg()	// op_1
readAndValidatePIN()	// op_1
readBankAccountDetailInfo()	// op_1
writePart1TransactionLonInfo()	// op_1
displayTransactionOptions()	// op_2
readSelectedTransactionOption()	// op_2
writePart2TransactionLogInfo()	// op_2
displayMaxWithdrawalAmount()	// op_3
displayEnterWithdrawalAmountMssg()	// op_3
readDesiredWithdrawalAmount()	// op_3
writePart3TransactionLogInfo()	// op_3
updateATMandBankAccountInfo()	// op_4
displayAccountDetailInfo()	// op_4
displayCollectCashMssg()	// op_4
dispenseCashOperation()	// op_4
writePart4TransactionLogInfo()	// op_4

Figure 4. AFS model (left) and class (right) for use-case U_1

C. AFS Model of U_1

Figure 4 shows $AFS(U_1)$, obtained by combining the DFD and the FSM shown in Figure 3. It also shows the SuccessfulWithdrawal-class obtained from $AFS(U_1)$ based on Tables I and II and the Class-Creation-Rules, and it is

suitable for implementing U_1. The detailed analysis of each t_j identified several low level functions (methods) for the associated op_j, and we have labeled each of those low-level functions as "//op_j" in SuccessfulWithdrawal-class. One could merge the functions with the same label "//op_j" into a single function, and call it op_j. If we let each merged function op_j have parameters corresponding to the dataflows to it and let each op_j call op_{j+1}, then we can eliminate all the class variables except d_1; even d_1 can be eliminated if we use a parameter in op_2 and pass d_1 for it when called by op_1. But we keep the class variables as shown to simplify the design of a single general purpose controller for any sets of use-cases.)

D. Merging and Decomposing Interactions in U_1

We can merge interactions t_1 and t_2 or, equivalently, states s_1 and s_2 in Figure 4 into a single state because $c_{1,2} = true$. This will not adversely affect handling U_2 and U_3 because both t_1 and t_2 are common to U_2 and U_3. We do not merge s_2 and s_3 because that would prevent sharing s_2 (and its associated operations, inputs, and outputs) between U_1 and the use-cases U_2 and U_3. On the other hand, we do not decompose a $t_j, 1 \leq j \leq 4$, into several $t_{j.k}$'s corresponding to the methods in Figure 4 having the label "//op_j" because that does not give us a better sharing of $t_{j,k}$'s among the use-cases $\mathcal{U} = \{U_1, U_2, U_3\}$, and thus does not help in the design of a class-structure for $\mathcal{U}$.

E. Classes and Formal Description of Use-Cases U_2 and U_3

Shown below are the decompositions of the use-cases U_2 and U_3 into their component interactions. The use-case U_2 shares its first three interations with U_1 and U_3 shares its first two interactions with U_1. Note that $t_{10} \in U_3$ is the same as $t_2 \in U_1$, even though the value of data-item $in^u_{10} = in^u_2$ is different in U_1 and U_3; that difference is reflected in $c_{2,3} \neq c_{10,11}$. The condition $c_{7,8} = \neg c_{3,4}$ gives $c_{7,8} \wedge c_{7,8} = false$ and $c_{3,4} \vee c_{7,8} = true$. The pre-condition for U_2 is $c_{2,3} \wedge c_{3,8}$ and that for U_3 is $c_{2,11}$.

U_2: Failed withdrawal due to insufficient funds

t_{j+4}: Same as **t_j** in U_1 for $1 \leq j \leq 3$.

t_8: User sees "insufficient funds" message. [**op_8:** ATM displays insufficient funds message for the desired withdrawal amount, and writes part-5 ("failed withdrawal", withdrawalAmount = 0) of TLI.]

U_3: Balance enquiry

t_9: Same as **t_1** in U_1.

t_{10}: Same as **t_2** in U_1 except that the user selects the "balance-enquiry" option.

t_{11}: User sees account balance information. [**op_{11}:** ATM displays the account balance and writes part-6 ("balance enquiry") of TLI.]

Table III gives the inputs and outputs for the interactions t_8, t_{10} and t_{11} in the use-cases U_2 and U_3. Table IV gives the conditions for these new interactions.

TABLE III. Inputs and outputs for U_2 and U_3 that are different from U_1

in_8	∅ (empty)	out_8	u: "Insufficient Funds" message e: Part-5 of TLI
$in_{10} = in_2$	u: Selected Transaction Option	$out_{10} = out_2$	∅ (empty)
in_{11}	∅ (empty)	out_{11}	u: Account Balance e: Part-6 of TLI

TABLE IV. Conditions $c_{j,k}$ for U_2 and U_3 that are different from U_1

$c_{7,8} = c_{3,8} = \neg c_{3,4}$	Max Allowable Withdrawal Amount < Withdrawal Amount Desired
$c_{9,10} = c_{1,2}$	true (no invalid Card)
$c_{10,11} = c_{2,11}$	"Balance Enquiry" = Selected Transaction Option

F. AFS Models for U_2 and U_3

We do not show the FSM and DFD for U_2 and U_3, but their AFS models and the corresponding classes are shown in Figures 5 and 6. As in the case of U_1, we could merge the functions (methods) with the same label "//op_j" in Figures 5 and 6 into a single function and call it op_j.

G. Merging of States for U_2 and U_3

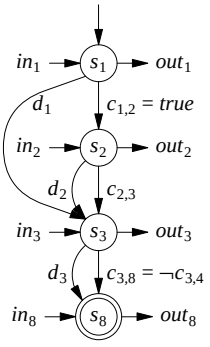
We do not merge s_2 with s_3 or s_{11} because $c_{2,3}$ and $c_{2,11}$ are disjoint; likewise, we do not merge s_3 with s_4 or s_8.

V. Class Structure and Implementation

The classes in Figures 4-6 together allow us to implement the ATM described in section IV-A. We get the class-hierarchy shown in Figure 8 when we eliminate the duplicate attributes and methods in these classes using refactoring and combine the classes into a hierarchy. We can also directly get the same class-hierarchy, without creating the classes in Figures 4-6, from the combined AFS model for U_1-U_3 shown in Figure 7.

A. Combining AFS Models

We use the following notion of *equivalent* states to combine two AFS models $M = AFS(\mathcal{U})$ and $M' = AFS(\mathcal{U}')$ for the sets of use-cases $\mathcal{U}$ and $\mathcal{U}'$. Two states $s_j \in M$ and $s'_j \in M'$ are equivalent if all computations along the path $\pi(s_j)$ from the start-state of M upto s_j are identical to those along the path $\pi'(s'_j)$ from the start-state of M' upto s'_j in terms of the

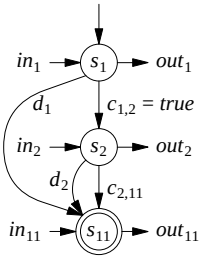


InsufficientFund	
ATMidAndOtherInfo	
bankDebitCardPINinfo	
bankDebitCardAccountDetailInfo	// d_1
selectedTransactionOption	// d_2
maxAllowableWithdrawalAmount	// d_3
desiredWithdrawalAmount	// d_3
readDebitCardPINinfo()	// op_1
displayEnterPINmssg()	// op_1
readAndValidatePIN()	// op_1
readBankAccountDetailInfo()	// op_1
writePart1TransactionLogInfo()	// op_1
displayTransactionOptions()	// op_2
readSelectedTransactionOption()	// op_2
writePart2TransactionLogInfo()	// op_2
displayMaxWithdrawalAmount()	// op_3
displayEnterWithdrawalAmountMssg()	// op_3
readDesiredWithdrawalAmount()	// op_3
writePart3TransactionLogInfo()	// op_3
displayInsufficientFundsMssg()	// op_8
writePart5TransactionLogInfo()	// op_8

Figure 5. AFS model and class for use-case U_2

order of computations and the underlying constraints. To be precise, s_j and s'_j are equivalent if

1) For $k < j$, s_k is equivalent to s'_k.
2) The dataflows to s_j and s'_j are identical in terms of the data-sources and the data-items.
3) The operations at s_j and s'_j are the same: $op_j = op'_j$, including the inputs $in^u_j = in'^u_j$ and $in^e_j = in'^e_j$ and the outputs $out^u_j = out'^u_j$ and $out^e_j = out'^e_j$.
4) The dataflows from s_j and s'_j are the same ($d_j = d'_j$).
5) The condition $c_{j-1,j}$ in M is the same as $c'_{j-1,j}$ in M', i.e., $c_{j-1,j} = c'_{j-1,j}$.



BalanceEnquiry	
ATMidAndOtherInfo	
bankDebitCardPINInfo	
bankAccountDetailInfo	// d_1
selectedTransactionOption	// d_2
readDebitCardPINinfo()	// op_1
displayEnterPINmssg()	// op_1
readAndValidatePIN()	// op_1
readBankAccountDetailInfo()	// op_1
writePart1TransactionLogInfo()	// op_1
displayTransactionOptions()	// op_2
readSelectedTransactionOption()	// op_2
writePart2TransactionLogInfo()	// op_2
displayAccountBalance()	// op_{11}
writePart6TransactionLogInfo()	// op_{11}

Figure 6. AFS model and class for use-case U_3

The conditions (1)-(3) and (5) above imply we can assume $d_j = d'_j$, i.e., condition (4) holds without loss of generality. This can be seen as follows. Because $op_j = op'_j$ can compute each of d_j and d'_j, it can compute $d_j \cup d'_j$ and thus we can replace each of d_j and d'_j by $d_j \cup d'_j$. The equivalence of s_j and s'_j depends only on the states on the paths $\pi(s_j)$ and $\pi'(s'_j)$, the inputs, outputs, and dataflows to and from those states, and the transition-conditions along $\pi(s_j)$ and $\pi'(s'_j)$. Note that the above definition of equivalence differs in many ways from that in finite-state automata theory, where the state-equivalence depends on what can happen in future from those states; in particular, the final states play a critical role. In our definition, the final-states have no special role.

B. Combining AFS models

Suppose U and U' are two (elementary) use-cases with one or more equivalent states. (We can always imagine a dummy start-state for a use-case, which just displays "starting..." and having no dataflow from this state. This will make the start-state of all use-cases equivalent.) We can merge the pairs of equivalent-states, one in each of $AFS(U)$ and $AFS(U')$, and the result is an AFS model having a tree-structure with two terminal nodes (final states). We can repeat the process for a set of use-cases $\mathcal{U}$, merging a state in $AFS(U_j)$ with its equivalent-state (if any) the result of merging $AFS(U_i), 1 \leq i < j$. The final AFS model $AFS(\mathcal{U})$ does not depend on the order in which we merge $AFS(U_i)$'s. Figure 7 shows the merged AFS model obtained from those in Figures 4-6.

The path $\pi = \langle s_1, s_2 \rangle$ in Figure 7 gives the class ATM-Transaction in Figure 8. The other classes in Figure 8 are obtained from the single-state paths $\langle s_3 \rangle$, $\langle s_4 \rangle$, $\langle s_8 \rangle$, and $\langle s_{11} \rangle$. The next-relationship between the paths $\langle s_1, s_2 \rangle$ and $\langle s_3 \rangle$ makes Withdrawal-class a subclass of ATMTransaction-class in Figure 8, and likewise for the other subclass-relationships. We obtain the same class structure in Figure 8 if we start with the classes in Figures 4-6 and apply refactoring [7].

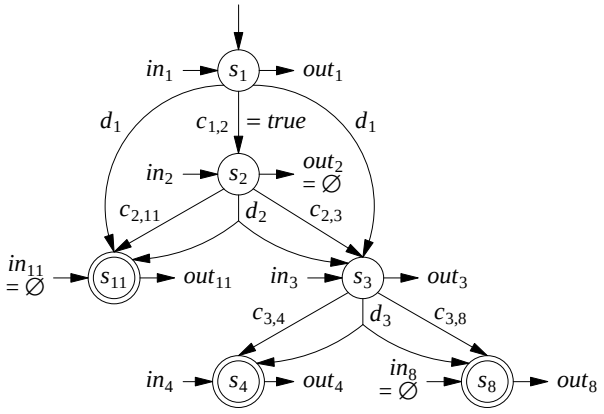


Figure 7. Combined AFS model for U_1-U_3.

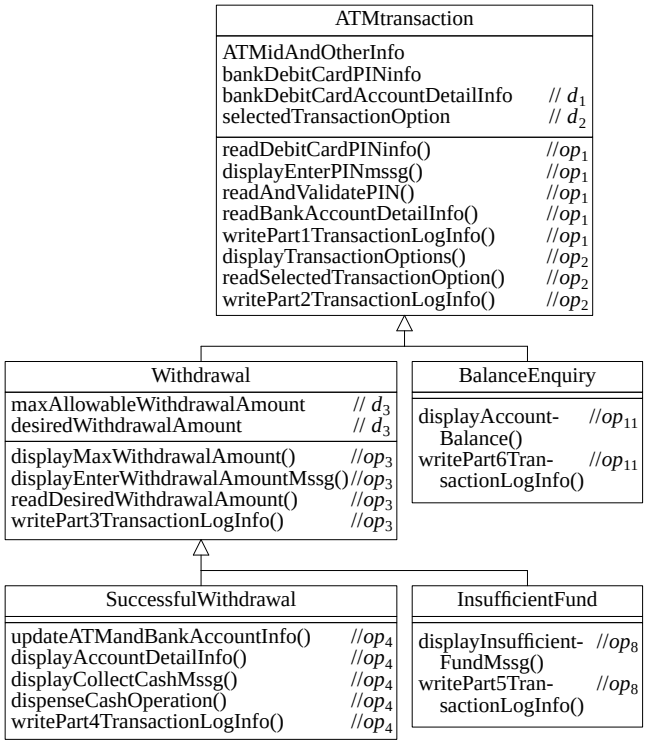


Figure 8. Class-hierarchy from the AFS model in Figure 7

It is worth pointing out that if we apply the concept analysis technique [12] to the attributes (variables) and methods of the classes in Figures 4-6, based on the use-relationship between those variables and methods, then we would arrive basically at the same final class structure in Figure 8, except that each of the classes ATMtransaction and Withdrawal will become a chain of simpler classes (involving a partitioning of variables and methods in those classes). We will then simply merge those chains to form the classes ATMtransaction and Withdrawal as given in Figure 8. Note that a major part of building $AFS(\mathcal{U})$ involves, via the details of the interactions in the use-case $\mathcal{U}$, the identification of all class variables and methods, and their use-relationships, and the latter are the inputs to concept analysis. The only part of $AFS(\mathcal{U})$ which does not explicitly appear in the class-structure and is not used in concept analysis

 ISBN: 978-1-61208-367-4

is the transition-conditions; they play, however, a critical role in the correctness of the tree-structure of $AFS(\mathcal{U})$, its unique decomposition into maximal linear segments (like $\langle s_1, s_2 \rangle$), and in developing the controller (see Section V-C) to drive the execution of the methods in the class-structure. In this sense, our approach based on the AFS model is superior to the concept analysis method. After all, there is not much value in a class-structure design unless we clearly understand how to control the calls to its methods.

C. Implementation

Figure 9 shows the *execution* dependencies among the methods for the AFS model in Figure 7. Here, f_j represents the group of methods corresponding to op_j (see Figure 8) for state s_j. A link (f_i, f_j) implies the execution of f_j follows that of f_i, partly because d_i generated by f_i is required by f_j for its computations or d_i is needed in determining whether f_j can execute or not. It is not surprising that the links (f_i, f_j) in Figure 9 parallel the links (s_i, s_j) in Figure 7.

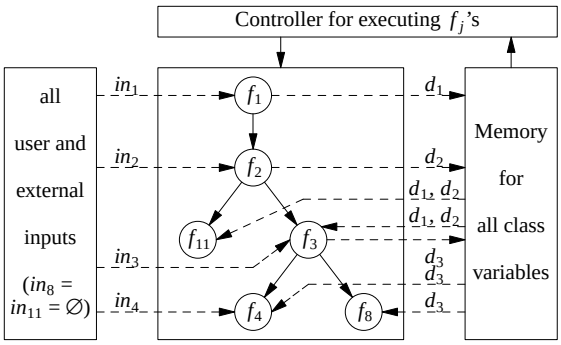


Figure 9. The execution dependency among methods f_j corresponding to op_j in Figure 8; d_j's are the dataflows.

There are many ways [13] to implement the dependencies in Figure 9. The simplest and the best solution is a central controller that works as follows. It first calls the root function f_1, and following the execution of an f_j it tests the disjoint conditions $c_{j,k}$ for the "child" functions f_k of f_j and calls f_k if $c_{j,k}$ is true. Another approach is to start with the root function f_1 as before but let each f_j evaluate the conditions $c_{j,k}$ and call f_k, if $c_{j,k}$ is true, as its last step, Here, changes in $\mathcal{U}$ may require small modifications to several f_j's, depending on how many classes are affected. In the first approach, the modifications to the central controller can be completely automated. As a third alternative, we can introduce parameters to f_j's and let each f_j store d_j as a local variable in it and use it as one of the parameters in the call to an f_k. For Figure 9. this means f_1 executes first and ends with a call to f_2, with d_1 as the parameter. Next, f_2 ends with a call to f_3 or f_{11}, with d_1 and d_2 as parameters. Likewise, f_3 ends with a call to f_4 with $d_1, d_2,$ and, d_3 as parameters, etc. This approach requires fewer class-variables, but changes in requirements may cause many changes in the definition and selection of the parameters. This approach clearly gives a poor quality software.

VI. Conclusion

We have presented here a systematic, semi-formal method to obtain a hierarchical class-structure, including the attributes and methods for each class in the hierarchy, for an object-oriented design of a software from its requirements given in the form of a set of use-cases. We use two formal models: (1) a detailed model of a use-case in terms of its interactions (operations), which includes the user and external inputs/outputs of each operation and the dataflows to/from it, and (2) an Augmented Finite State (AFS) model for a set of use-cases, which captures shared operations among the use-cases, the points-of-divergence between use-cases and the related control-flow conditions. The AFS model can help to identify missing use-cases, missing interactions in the use-cases, and the potential need to decompose some interactions into simpler ones. If the AFS model has a tree structure, then this directly gives a hierarchical class-structure suitable for an object oriented implementation of the requirements. This means, in principle, one could start from, say, a C-program P and reverse engineer it to obtain an AFS model for it, and if the AFS model has a tree structure then create a hierarchical class-structure from that AFS model, and finally obtain an object-oriented program P' with the same functionality as P.

Acknowledgment

The authors would like to thank the anonymous referees for some useful comments to improve the presentation.

References

[1] D. K. Deeptimahanti and R. Sanyal, "Semi-automatic generation of uml models from natural language requirements," *Proc. of the 4th India Software Engineering Conference*, pp. 165–174, 2011.

[2] L. Li, "Translating use cases to sequence diagrams," *Proc. of 15th IEEE Intern. Conf. on Automated Software Engineering*, pp. 293–296, 2000.

[3] D. Liu, K. Subramaniam, B. Far, and A. Eberlein, "Automating transition from use-cases to class model," *IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, vol. 2, pp. 831–834, 2003.

[4] G. S. A. Mala and G. V. Uma, "Automatic construction of object oriented design models [uml diagrams] from natural language requirements specification," *Proc. of 9th Pacific Rim Intern. Conf. on Artificial Intelligence*, pp. 1155–1159, 2006.

[5] A. Cockburn, "Structuring use cases with goals," *Journal of Object-Oriented Programming*, 1997.

[6] B. Roussev, "The value added invariant: A newtonian approach for generating class diagrams from a use case model," *WITUML, 16th European Conf. on Object Oriented Programming, ECOOP-2002*, 2002.

[7] M. Fowler, K. Beck, J. Bryant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[8] S. Kundu, "Structuring software functional requirements for automated design and verification," *Proc. 31st Annual IEEE Intern. Computer Software and Applications Conf, COMPSAC-07*, Jul 24-27, 2007.

[9] T. Chow, "Testing software design modeled by finite-state machines," *IEEE Trans. Softw. Eng.*, vol. SE-4, no. 3, pp. 178–187, 1978.

[10] S. Eilenberg, *Automata, Languages, and Machines*. Orlando, FL, USA: Academic Press, Inc., 1976.

[11] I. Jacobson, *Object-Oriented Software Engineering: A Use Case Driven Approach*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 2004.

[12] G. Snelting and F. Tip, "Understanding class-hierarchies using concept analysis," *ACM Trans. Prog. Lang. Syst.*, vol. 22(3), pp. 540–582, 2000.

[13] S. Kundu, "A canonical functional design based on the domination-relationship among data," *Proc. 8th Asia Pacific Software Engineering Conference, APSEC-2001*, Dec. 4-7, 2001.

 ISBN: 978-1-61208-367-4

Fundamentals, Prospects and Challenges for Totally Functional Programming Style

Paul Bailes, Leighton Brough, Colin Kemp
School of Information Technology and Electrical Engineering
The University of Queensland,
St Lucia, QLD Australia
{p.bailes, l.brough, c.kemp}@uq.edu.au

***Abstract*—General recursive definitions contribute to the complexity of programming. This complexity could be reduced by reliance on established, well-understood programming patterns. Catamorphism-based recursion patterns simplify programming with little practical loss in expressive capability compared to general recursion, including the capability of defining new recursion patterns. Partial application of catamorphisms, sub-catamorphic recursion patterns and methods to symbolic data allows a comprehensive replacement of symbolic data with functional, or what we describe as "zoetic", representations that inherently adopt the benefits of catamorphism-based programming. The considerable promise of this "Totally Functional" style confronts us with some exciting technical challenges.**

Keywords-component; Catamorphism, Fold, Functional, Recursion.

I. INTRODUCTION

We contend that software is unduly complicated by the pervasive need to program interpreters for the computations inherent to symbolic data. By using instead functional representations that embody the fusion of characteristic interpretations into data, this pervasive complication can be minimized if not avoided, and programming thus significantly simplified.

Our essential argument develops in logical sequence as follows:

- recursion patterns such as list "foldr", which generalize to catamorphisms on regular recursive datatypes, suffice to express and simplify a very wide range of common recursive definitions;
- other useful and simplifying recursion patterns are also definable in terms of catamorphisms;
- catamorphisms thus embody practically as well as theoretically (in terms of initial algebra semantics) the behaviours characteristic to abstract data types;
- partial application of catamorphisms to typical symbolic representations of data yield functional representations that inherently possess these characteristic behaviours, i.e., a kind of liveness which we describe as "zoetic" from the Greek "zoion" meaning "animal" (as in "zoology");
- partial application of behaviours that are more specialized than the generic catamorphism, but are definable inevitably in terms of catamorphisms, also yield zoetic data;
- programming with zoetic data simply involves their application to appropriate operands (just as with recursion patterns), rather than also having to program with explicit recursion the characteristic behavior of the datatype;
- creation of zoetic data can be effected by generator functions which are the derived counterparts of symbolic data constructors;
- this enables a new style of programming (which to emphasise its distinctiveness from an earlier related development of "Total Functional Programming" [1] we call "Total**ly** Functional Programming", or TFP), in which a comprehensive supercession of symbolic data by functional representations can be achieved;
- while the comprehensiveness of the foregoing program is unprecedented, important aspects of it are discernable in (and thus validated by) related fields of computer science.

The presentation of the argument in this paper follows the above sequence.

The consequent comprehensive replacement of symbolic data by functions requires first-class functions, hence we implicitly adopt functional programming [2] and functional languages. We choose Haskell [3] for purposes of illustration.

II. CATAMORPHIC PROGRAMMING

An approach to programming based entirely on canonical recursion patterns known as "catamorphisms" [4] is beneficial, viable and self-sufficient. Catamorphisms are more familiar as the list "reduce" or "foldr" functions of functional programming, but apply to all regular recursive types.

A. General Recursion Too Complex

Recursion patterns simplify and clarify programming, compared to the use of general iteration/recursion. Consider the case of recursively defining basic arithmetic operations on the simplest recursive datatype, of Natural numbers, in Fig. 1. This example exposes some key aspects of Haskell as follows:

- declaration of datatypes (e.g., Nat) in terms of constructor functions (e.g., Zero and Succ) and their operand types where appropriate (i.e., Nat, thus defining a recursive type);

 ISBN: 978-1-61208-367-4

```
data Nat = Zero | Succ Nat

add Zero b = b
add (Succ a) b = Succ (add a b)

mul Zero b = Zero
mul (Succ a) b = add b (mul a b)

exp a (Zero) = Succ Zero
exp a (Succ b) = mul a (exp a b)
```

Figure 1. General recursive renditions of basic arithmetic operations.

```
cataN Zero f x = x
cataN (Succ n) f x = f (cataN n f x)

add a b = cataN a Succ b
mul a b = cataN a (add b) Zero
exp a b = cataN b (mul a) (Succ Zero)
```

Figure 2. Catamorphic renditions of basic arithmetic operations.

- definition of functions by recursion equations;
- branching by pattern matching on function arguments;
- function application by juxtaposition of operator and operand(s).

(Further key novelties of functional languages and Haskell in particular will be explained as introduced in examples below.)

In this general recursive rendition of arithmetic operations, the following deficiencies are apparent.

Apart from the suggestive naming of the type and its constructors, there is nothing in the definition that compels treatment of members of the type as naturals, or indeed numbers of any kind;

Instead, the obvious isomorphism between the concrete members of Nat and the abstract natural numbers needs to be implemented by an implicit interpreter that converts symbols into actions (in this case, iterative applications of other functions);

A programmer needs to repeat the implementation of this interpreter at each usage of Nat entailing not just extra effort but the risk of inconsistent implementations leading to inconsistent (erroneous) behavior.

Using however the "catamorphism" recursion pattern on Nat - cataN - the rendition becomes that of Fig. 2 which significantly remedies the above deficiencies, in that a uniform interpretation of the symbolic data is provided - i.e., cataN - which moreover derives directly from the type definition.

B. Catamorphisms as Practical Basis

The catamorphic pattern defined on Nat above generalises for regular recursive types. For example, the catamorphism - cataL - for (polymorphic) lists is as in Fig. 3. Note how in Haskell the type polymorphism on type List is signified by parameterization on list element type 't'.

```
data List t = Cons t (List t) | Nil

cataL Nil o b = b
cataL (Cons x xs) o b = o x (cataL xs o b)
-- versus
foldr op b [] = b
foldr op b (x:xs) = op x (foldr op b xs)
```

Figure 3. Catamorphisms and operations on lists.

```
sumR Nil = 0
sumR (Cons x xs) = x + sumR xs
-- versus
sumC xs = cataL xs (+) 0

appendR Nil ys = ys
appendR (Cons x xs) ys =
   Cons x (appendR xs ys)
-- versus
appendC xs ys = cataL xs Cons ys
```

Figure 4. List processing examples.

Observe also that (aside from a reordering of the usual presentation of operands) cataL is exactly the familiar "foldr" of functional programming (also known as "reduce").

The reader will also observe that just as with Nat above, operations on lists may be programmed using the uniform interpretation offered by cataL applied to other operations and data pertaining to the specific applications. See Fig. 4 for a comparison of explicit recursive vs. catamorphic definitions of some basic list processing functions. (Note how in Haskell the form "(θ)" denotes the function represented by operator 'θ', in this case binary addition represented by '+'.)

What make catamorphisms attractive as a practical as well as a theoretical basis for programming are their properties as follows:

- generality - existence for all regular recursive types, not just Nats or Lists
- expressiveness - sufficient to define at least any function provably-terminating in second-order arithmetic [5], i.e., practically-speaking any reasonable function other than a Universal Turing Machine or other programming language interpreter;
- essentiality - their embodiment of the initial algebra semantics [6] of the respective underlying datatypes, as the unique homomorphisms that define the applicable notion of initiality itself;

C. Catamorphisms as Pragmatic Basis

There are however other recursion patterns that appear to be necessary for the natural solution of programming problems. For example, compare the catamorphic renditions in Fig. 5 of the "insert" and "reverse" operations with their definitions in Fig. 6 using respectively the paramorphic [4] and "left fold" [2] recursion patterns. (N.B. our adoption henceforth of customary concrete syntax for the List type.)

 ISBN: 978-1-61208-367-4

```
-- insert element into ascending list
insert e xs =
  fst $ cataL xs
  (\x (exs,xs) ->
    (if e<x then e:x:xs else x:exs, x:xs)
  )
  ([e],[])

-- reverse order of list elements
reverse xs =
  cataL xs
  (\x xs' -> (\rxs -> xs' (x:rxs)))
  (\rxs -> rxs)
  []
```

Figure 5. Catamorphic definitions of "insert" and "reverse".

```
insert e xs =
  paraL xs
  (\x exs xs ->
    if e<x then e:x:xs else x:exs
  )
  [e]

reverse xs = lfold xs (\rxs x -> x:rxs) []

-- definitions of new recursion patterns

-- like cataL but op also has list tail xs
paraL (x:xs) o b = o x (paraL xs o b) xs
paraL [] o b = b

-- like cataL but op grouped from left
lfold (x:xs) o b = lfold xs o (o b x)
lfold [] o b = b
```

Figure 6. Alternative definitions of "insert" and "reverse".

```
paraL xs o b =
  fst $ cataL xs
  (\x (pxs,xs)->(\o b->(o x pxs xs, x:xs))
  (b, [])

lfold xs o b =
  cataL xs
  (\x lxs -> (\b -> lxs (o b x)))
  (\b -> b)
  b
```

Figure 7. Catamorphic definitions of other recursion patterns.

Important new Haskell features used here are as follows:

- anonymous "lambda" functions, of the form `(\ arguments -> body)`
- built-in list datatype, with constructors ':' (for Cons) and "[ ]" (for Nil);
- n-tuple data, with elements selected by pattern-matching or by selector functions (e.g., "fst");
- low-precedence function application denoted by '$'.

What allows us to continue to treat catamorphisms as a basis in the face of the above is that these other recursion patterns can be synthesized from catamorphisms without recourse to general recursion. The recursion patterns (such as paraL, lfold, etc.) can be defined using abstractions (higher-order, as needed) from the definitions of the methods (such as insert, reverse, etc.), e.g., as in Fig. 7.

D. Recursion Pattern/Application Hierarchy

The consequence of the above is that all the reasonable methods (on regular recursive datatypes, such as natural numbers, lists, trees, etc.) we would want to program, and all the recursion patterns besides catamorphisms that we would want to use to program them, can be derived in a hierarchical manner, starting from catamorphisms and supplying instantiating operands at each level of refinement.

For example, for lists:

- the root, catamorphism level of the hierarchy is represented by cataL;
- the intermediate, recursion pattern level of the hierarchy is represented by patterns derivable from the root, e.g., paraL; lfold; etc.;
- the lowest, application level of the hierarchy is represented by actual list operataions, e.g., length; append; insert; reverse; etc.

Note that members of the hierarchy at all levels are directly accessible from the root catamorphism, in some cases more conveniently (e.g., length) and in some cases less so when one of the intermediate recursion patterns is more convenient (e.g., reverse). In particular, the identity property of catamorphisms is that application of the catamorphism for a type to the constructors of that type returns the catamorphism, e.g., as follows:

```
cataN n Succ Zero = n
cataL xs (:) [] = cataL xs
```

III. ZOETIC DATA

The foregoing catamorphism-based recursion pattern approach to programming enables us to bypass completely symbolic data, and their interpretation. In the end, we directly construct "zoetic" representations of data, i.e., as functions, rather than the usual symbolic forms.

A. Catamorphic Zoetic Data (CZD)

CZD are the basic kind of zoetic data. They are formed by the partial application to symbolic values of the standard interpretation of their datatypes. The standard interpretation of each datatype remain exactly as exposed above, i.e., its catamorphism. As a result, each CZD is a function that implements that catamorphism on the underlying symbolic datatype.

Thus, the advantage of CZD is that their usages no longer require any interpretations as reflected by explicit recursive definitions or by the explicit application of the interpreter for the type, i.e., its catamorphism. Instead, the CZD are simply applied to appropriate catamorphism operands. For example, compare the above definitions of arithmetic operations to

```
-- zoetic naturals
zero = cataN Zero
one = cataN (Succ Zero)
-- etc

-- zoetic arithmetic operations
addz za zb = za succ zb
mulz za zb = za (addz zb) zero
expz za zb = zb (mulz za) one
```

Figure 8. Zoetic natural numbers and operations.

those on natural number CZD as in Fig. 8. Observe how zoetic naturals are simply the partial applications of cataN to the symbolic values of type Nat.

A final key Haskell feature found in the above is partial application of "curried" functions. For example, addz can equally be thought of as a function of one parameter "za" that returns a function of a further parameter "zb", as well as a function of the same two parameters. Thus, e.g., application of addz to the "zb" parameter of mulz denotes a function that will add "zb" to its further actual parameter.

Now, we can define generators, i.e., the zoetic counterparts of symbolic data constructors but independent of them. For example, from specifications as in Fig. 9, we derive the respective zoetic counterparts zero and succ of Zero and Succ as in Fig. 10 (likewise for cons and nil). Note that the identity property for catamorphisms and symbolic data constructors applies for CZD and their generators, e.g., as per the identities as in Fig. 11.

```
-- generally
zn = cataN n

-- specifically
zero s z = cataN Zero -- as above
succ (cataN n) = cataN (Succ n)
```

Figure 9. Specifications of zoetic natural number generators.

```
zero f x
= cataN Zero f x
= x

succ zn f x
= succ (cataN n) f x
= cataN (Succ n) f x
= f (cataN n f x)
= f (zn f x)

-- similarly derivable
cons x zxs o b = o x (zxs o b)
nil o b = b

-- etc. for other types
```

Figure 10. Zoetic natural number generators.

```
succ zn succ zero = succ zn
zero succ zero = zero

cons z zxs cons nil = cons z zxs
nil cons nil = nil
```

Figure 11. Identities for CZD.

Just as with zoetic arithmetic, zoetic list processing also entails simple, non-interpretive provision of relevant catamorphic operands, e.g., as follows.

```
zappend zxs zys = zxs cons zys
zsum zxs = zxs addz zero
```

B. Subcatamorphic Zoetic Data (SZD)

The interpretation of symbolic data is not always given by a catamorphism, but maybe by some other method that can be defined catamorphically, i.e., found below catamorphisms in the recursion pattern/application hierarchy, hence "subcatamorphic".

For example, in Fig. 12 the method "memb" interprets binary trees as sets, with constructors Nd, Lf and Tip respectively signifying set union, singleton and empty sets. However, within the catamorphic programming paradigm essential to TFP, these other methods (exemplified here by memb) will be expressible as catamorphisms, e.g., as in Fig. 13.

As with CZD, we form zoetic data by the partial application to the symbolic data of the interpreter for the required characteristic behaviour. In this case, the partial application "memb bt" (for some bt :: Bt) yields a function that tests if some putative element e is actually a member of the set represented by bt. That is, the SZD form of a set is the familiar characteristic predicate representation.

```
data Bt t = Nd (Bt t) (Bt t) | Lf t | Tip

memb (Nd t1 t2) e = memb t1 e || memb t2 e
memb (Lf x) e = x==e
memb Tip e = False
```

Figure 12. Trees as sets.

```
memb s e =
  cataBt s
    (\t1' t2' -> t1' || t2')
    (\x -> x==e)
    False

-- catamorphism on Bt
cataBt (Nd t1 t2) n l t =
  n (cataBt t1 n l t) (cataBt t2 n l t)
cataBt (Lf x) n l t = l x
cataBt Tip n l t = t
```

Figure 13. Set membership as a catamorphism.

 ISBN: 978-1-61208-367-4

```
memb s e =
  cataBt s
    (\s1 s2 e -> s1 || s2)
    (\x e -> x==e)
    (\e ->False)
    e
```

Figure 14. Catamorphic set membership with closed terms.

```
memb s =
  cataBt s
    (\s1 s2 e -> s1 || s2)
    (\x e -> x==e)
    (\e ->False)
```

Figure 15. Catamorphic set membership as a characteristic predicate.

```
union s1 s2 e = s1 e || s2 e
single x e = x==e
empty e = False
```

Figure 16. Declarations of generators for zoetic sets.

Further, if we write the catamorphism operands as closed terms as in Fig. 14, then, as a corollary of the identity property of catamorphisms, these closed operands serve as generators of characteristic predicates.

First, eta-reduction of the definition of memb exposes the zoetic set/characteristic predicate clearly as in Fig. 15. Then recognizing that the significance of the identity property is that catamorphism operands serve as constructor replacements, we see that catamorphism operands are inherently generators of whatever is the result of the catamorphism, in this case the zoetic set. So, finally rewriting the above operands in more convenient equational format gives the generator declarations of Fig. 16. The same technique applies for any SZD, subject of course to the condition that the characteristic method is definable as a catamorphism (which as we have seen is practically always).

C. Recursion Patterns as SZD

Just as applications such as "memb" give rise to SZD, so do the recursion patterns found below catamorphisms. For example, partial applications of the form "lfold xs" give rise to a class of list-like SZD, but which instead of having the catamorphic/foldr behavior of list CZD, behave as "left folds" with the binary operator 'o' grouped from the left.

Further, just as with zoetic sets above, when we express the catamorphic definitions of recursion patterns with closed operands e.g. as in Fig. 17, these operands are also effective as generators. Continuing the example, first eta-reduce as in Fig. 18 which exposes the zoetic left-folding list as an identity between a partial application of the lfold method and a catamorphism. Then we simply read off the operands to the catamorphism and re-present them as generator declarations as in Fig. 19.

```
lfold xs o b =
  cataL xs
  (\x lxs -> (\o b -> lxs o (o b x)))
  (\o b -> b)
  o b
```

Figure 17. Left fold as a catamorphism with closed terms.

```
lfold xs =
  cataL xs
  (\x lxs -> (\o b -> lxs o (o b x)))
  (\o b -> b)
```

Figure 18. Left fold as catamorphism partial application.

```
lcons x lxs o b = lxs o (o b x)
lnil o b = b -- NB same as nil CZD
```

Figure 19. Declarations of generators for left-folding zoetic lists.

D. Zoetic Data Hierarchy

The hierarchy of zoetic data (CZD and SZD) naturally parallels that of catamorphisms, other recursion patterns, and catamorphic methods as detailed above, in which descent in the hierarchy from most general CZD to more specialized SZD in achieved by application to appropriate operands.

For example, from a zoetic list zxs we can first derive the variant lzxs with the same elements in the same sequence but with left-folding behavior, by applying zxs to the left-folding zoetic list generators thus:

```
lzxs = zxs lcons lnil
```

Next, we can calculate the reverse of zxs by applying lzxs to the appropriate operands to left-fold as follows:

```
rzxs = lzxs (\rzxs x -> cons x rzxs) nil
```

Note that the zoetic nature of the resulting list is achieved by use of the zoetic list generators cons and nil in the above, rather than symbolic list constructors (:) and [ ]. That is, if conventional lists were the required result, we would have written instead the following:

```
rxs = lzxs (\rxs x -> x:rxs) []
```

(Further note how in this case the operands to the left-folding zoetic list lxzs are precisely those given to lfold above in order to reverse a conventional list.)

If desired, we can define a self-contained reverse operation, by application of successive sets of (sub-) catamorphism operands in stages reflective of the above, as per Fig. 20. A one-stage definition of zrev in Fig. 21 echoes the direct definition of list reversal as a catamorphism further above. This version however loses some of the transparency of the two-stage definition that results from being able to express zrev in its more natural left-folding form.

Finally, it one exists, we can always recover the symbolic form of a zoetic datum by applying it to the symbolic constructors, e.g., as in Fig. 22.

 ISBN: 978-1-61208-367-4

```
zrev zs =
  -- start with zs
  zs
  -- next transform into left-folding list
  lcons lnil
  -- finally give left-folding operands
  (\rzxs x -> cons x rzxs) nil
```

Figure 20. Staged defintion of list reverse.

```
zrev zs =
  zs
  (\x zxs'->(\rzxs -> zxs' (cons x rzxs)))
  (\rzxs -> rzxs)
  nil
```

Figure 21. Direct catamorphic definition of list reverse.

```
-- an identity, not a function definition
cons 'a' (cons 'b' nil) (:) [] = "ab"
```

Figure 22. Recovery of symbolic from zoetic data.

IV. TOTALLY FUNCTIONAL PROGRAMMING

Supported by the techniques presented above, our key key proposition in Totally Functional Programming (TFP) is the combination of three complementary factors.

First, every data type has a characteristic behavior (for pure structures such as naturals, lists, trees, etc. it is their catamorphism; for more specialised types it is the characteristic method for the type e.g., for sets it is the memb(er) function);

Second, the complexity of conventional programming derives significantly from the need to program the interpretation of these inherent behaviours from symbolic datatypes, which are typically intertwined with application-specifics (e.g., the explicit recursive definitions of arithmetic operations far above);

Finally, direct zoetic representations of data that embody these behaviours are specified as the partial applications of characteristic methods to the symbolic representations, and can be exploited simply by application to the further operands of the methods. The feasibility of TFP is enhanced by direct generation of CZD and SZD without having explicitly to apply the characteristic methods to symbolic data.

A potential criticism of the key proposition of TFP is that whereas it posits a single behavior or characteristic method for each datatype, instead multiple methods are normal in programming. Our response is that the zoetic data hierarchy for each type adequately expresses any need for multiple behaviours: at the summit of the hierarchy is the catamorphism, from which all other behaviours can be derived; more specific behaviours can be found lower in the hierarchy. The designer of a zoetic datatype is thus free to choose a relatively general (= more methods) or specific (= fewer methods) behavior as circumstances require.

V. RELATED WORK

Aside from our own work (recently [7][8][9]), some aspects of TFP have been presaged (and therefore in a sense pre-validated) by others. However, none of these propose the comprehensive replacement of symbolic data with functional/zoetic representations as we do.

A. TFP in Functional Programming

Our conception of TFP can already be discerned in various aspects of functional programing: Church numerals [10] are CZD for the Nat type above; combinator parsers [11] are SZD for context-free grammars.

B. Turner's Total Functional Programming

Turner's already-cited [1] related conception of TFP has a common basis with ours in the avoidance of general recursion in favour of recursion patterns such as catamorphisms (and additionally anamorphisms - see below), but does not eschew symbolic data as we do.

C. Language Design

The history of language design can be thought of as a progressive retrofitting of "Church" concepts into a "Turing" context. TFP culminates that process by the complete replacement of Turing-style interpretation of symbolic data with Church-style direct definitions of (higher-order) functions. Some highlights of this process with particular relevance to TFP are as follows (in reverse chronological order).

Backus [12] repudiated general recursion for a fixed set of "combining forms" (including list catamorphisms), but without generalization to other types.

Dijkstra's [13] emphasis on fixed control structures rather than arbitrary control flows ("goto" statements) can be thought of a similar in sprit to our (and Turner's and Backus') repudiation of general recursion.

But long before, Backus equipped FORTRAN with the catamorphism on natural numbers, in the form of the DO-loop. Our TFP of course offers the programmer significantly more facility than DO-loop programming.

It is evident from this paper that modern functional languages (such as Haskell) at least encourage TFP. However, in order to avoid surprising limitations on zoetic operations, it will be necessary to adopt more powerful type systems (see Future Directions/Type-checking below). Also, in order to dispense with symbolic data completely, it will be necessary to handle infinite structures zoetically (see Future Directions/Codata and Corecursion below).

VI. FUTURE DIRECTIONS

TFP's promise is also a stimulus to address some key technical challenges, in the following respects.

With respect to computer science education: the simplicity of recursion-pattern-based programming (no need to program iteration or recursion; just "complete the blanks" by supplying catamorphic recursion patterns with the appropriate operands as in the examples above) suggests applicability to introductory programming teaching.

 ISBN: 978-1-61208-367-4

```
anL nxt end seed =
   if end seed then []
   else
      let (nxtelt, nxtseed) = nxt seed
      in nxtelt : anL nxt end nxtseed
evens =
  anL (\s->(s+2,s+2)) (\_->False) 0

odds =
  anL (\s->(s+2,s+2)) (\_->False) 1

fibs =
  anL
  (\(fa,fb)->(fa,(fb,fa+fb)))
  (\_->False)
  (0,1)
```

Figure 23. Anamorphic defintions of infinite streams.

Regarding type-checking: the convenient type inference found in Haskell and other modern functional languages does not accept some simple CZD (arithmetic on Church numerals). It's not yet clear if the existing candidates for the necessary more complex type systems are prohibitively inconvenient [14].

Regarding formal methods: just as catamorphisms (and CZD) possess more specific (and useful) laws than induction [15], what kind of more specialised laws are derivable among more specialised zoetic data (i.e., SZD)? With respect to course code refactoring: if zoetic data represent a clearer way to write programs, they should equally represent a good refactoring target, as indicated by some potentially useful results already [16][17].

Finally, regarding processing of infinite structures: catamorphisms are total functions on finite structures ("data"), but for practical computing, processing ("corecursion") of (potentially) infinite structures ("codata") is clearly necessary (e.g., a stream of transactions against a database; events to which a real-time operating system has to respond; etc.). The clear path to a solution [1] entails "anamorphisms" [4], as the categorical dual to catamorphisms, to provide the effective basis for zoetic representations of codata. For example, the anamorphism on lists ("anL" a.k.a. "unfold") can be used to define (infinite) streams, as in Fig. 23. Note that in Haskell, the '_' denotes an ignorable formal parameter, useful in defining constant functions. We are however yet to develop a presentation of anamorphism-based zoetic codata in the same comprehensive way that we have achieved for catamorphism-based zoetic data.

VII. CONCLUSIONS

Totally Functional Programming has the promise to fulfill the prospects of functional programming in several ways. Fundamentally, the essence of functional programming - "first class" functions - is exploited to simplify programming by bypassing pervasive interpretation with zoetic data that encapsulate the behaviours essential to data.

Higher-order functions are also instrumental in realizing the pragmatics of TFP - for each type, a hierarchy from general (CZD) to specific (SZD) entities exists, the specialization relationship being implemented by application to catamorphism operands.

Finally, as signified by the "front of stage" role it gives to zoetic data (compared to their hitherto relegation as theoretical curiosities as "Church" data representations), TFP completes an important stage in the last sixty or so years of work of restoring the Church perspective programming into the otherwise Turing-dominated worldview.

REFERENCES

[1] D. A. Turner, "Total Functional Programming", J. Universal Computer Science, vol. 10, no. 7, 2004, pp. 751-768.

[2] J. Hughes, "Why Functional Programming Matters", The Computer Journal, vol. 32, no. 2, 1989, pp. 98-107.

[3] The Haskell Programming Language, http://www.haskell.org/haskellwiki/Haskell, retrieved: 11 August 2014.

[4] E. Meijer, M. Fokkinga, and R. Paterson, "Functional Programming with Bananas, Lenses, Envelopes, and Barbed Wire", Proc. FPCA 1991, LNCS vol. 523, 1991, pp. 142-144.

[5] J. Reynolds, "Three approaches to type structure, Mathematical Foundations of Software Development", LNCS vol. 185, 1985, pp. 97-138.

[6] R. Backhouse, P. Jansson, J. Jeuring, and L. Meertens, "Generic Programming - An Introduction", in S. Swierstra, P. Henriques and J. Oliveira (eds.), Advanced Functional Programming, LNCS, vol. 1608, 1999, pp. 28-115.

[7] C. Kemp, "Theoretical Foundations for Practical 'Totally-Functional Programming' ", PhD Thesis, The University of Queensland, St Lucia, 2009.

[8] P. Bailes and L. Brough, "Making Sense of Recursion Patterns", Proc. 1st FormSERA: Rigorous and Agile Approaches, IEEE, 2012, pp. 16-22.

[9] P. Bailes, L. Brough, and C. Kemp, "Higher-Order Catamorphisms as Bases for Program Structuring and Design Recovery", Proc. IASTED SE, 2013, pp. 775-782.

[10] H. Barendregt, The Lambda Calculus - Its Syntax and Semantics 2nd ed., North-Holland, Amsterdam, 1984.

[11] G. Hutton, "Higher-order functions for parsing", Journal of Functional Programming, vol. 2, 1992, pp. 323-343.

[12] J. Backus, "Can programming be liberated from the Von Neumann style? A functional style and its algebra of programs", Comm. ACM, vol. 9, 1978.

[13] E. Dijkstra, "Goto Statement Considered Harmful", Comm. ACM, vol. 11, 1968, pp. 147-148.

[14] D. Vytiniotis, S. Weirich, and S. L. P. Jones, "Boxy types: inference for higher-rank types and impredicativity", Proc. ICFP 2006, 2006, pp. 251-262.

[15] G. Hutton, "A Tutorial on the Universality and Expressiveness of Fold", Journal of Functional Programming, vol. 9, 1999, pp. 355-372.

[16] J. Launchbury and T. Sheard, "Warm Fusion: Deriving Build-Catas from Recursive Definitions", Proc. FPCA 1995, ACM, New York, 1995, pp. 314-323.

[17] S. Mak and T. van Noort, Recursion Pattern Analysis and Feedback, Center for Software Technology, Universiteit Utrecht, 1986.

 ISBN: 978-1-61208-367-4

Using Automatic Code Generation Methods for Reusable Software Component Development: Experience Report

Elif Kamer Karataş, Barış İyidir
Defense System Technologies Division
Aselsan
Ankara, Turkey
{ekkaratas, biyidir}@aselsan.com.tr

***Abstract*— Quality of reused components becomes one of the dominating factors on the overall quality of the software when the component-based approach is adopted for development. In cases where reusable components are developed to be compatible with reference architecture, the contracts of the components are predefined. Nevertheless, the detailed design and implementation of the component depends mostly on the experience of the developers. The quality and the productivity of component development process can be improved by systematic sharing of domain knowledge and experiences. In this paper, automatic code generation is adopted in order to achieve systematic distribution of this knowledge throughout developers. Also, the experiences gained during the application of automatic code generation approach for the development of components that communicate via serial channel protocols are shared.**

Keywords-code generation; domain specific languages; domain knowledge

I. INTRODUCTION

The proposed automation method is aimed to be used in Embedded Real-time Control Software (ERCS), which is mission critical software that collects data from its sensor environment and processes them with control algorithms to give the proper commands to its actuation environment. The quality of software in such systems is of great importance since the cost of any failure is very high.

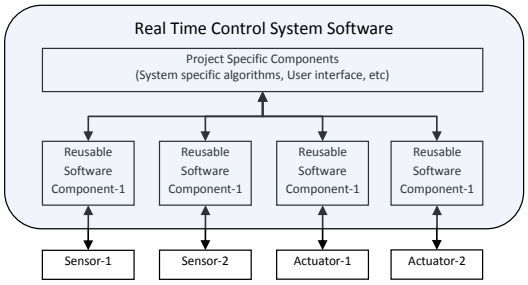


Figure 1. Real-Time Control System & Software Architecture.

The real-time control system architecture and the corresponding layered software architecture are given in Figure 1. The reusable software components, which are the main candidates for automation in this study, are responsible for the communication with the surrounding sensors and actuators to receive data and to give commands. These components are developed according to Interface Control Documents (ICDs) delivered together with the sensor or actuator hardware.

In ERCS software, the analysis of defects showed that the average ratio of errors originating from reusable components is 23.48%, which is the primary motivation of quality improvement studies on these components. Difficulties in sharing domain rules and experiences with developers and also the difficulties in proving the conformance to such rules are regarded as significant obstacles on the way to improve the quality and productivity.

Generative programming is defined as a class of tool technology that captures knowledge about how to generate code by enabling automation [1]. Generators are usually based on domain specific notations and they close the gap between high-level system description and executable [2]. Since auto generated codes enforce domain rules and best practices, they provide an effective way for uniformly sharing of domain knowledge among development teams. This study aims the systematic distribution of domain knowledge and expertise using automatic code generation methods.

The paper is organized as follows: Section 2 gives a brief literature review. The proposed method for automatic code generation is described in Section 3. Case studies and the results obtained are given in section 4. Section 5 discusses the results and proposes future works. Starting from section 3 of this text the term "component" will refer to mention reusable components in ERCS.

II. LITERATURE

Software development began with the employment of low level binary machine language and went along with the introduction of assembly language, high-level languages, modeling languages and Domain Specific Languages (DSLs) for providing increased abstraction for developers while performing the generation of source-code automatically [3]. Although code generation without modifications by developer seems impossible, developers are exempted from writing large amounts of repetitive or trivial code sections and have more time to focus on their core engineering problems [3][4].

The advantages of automatic code generation mentioned by Cullum [5] are:

 ISBN: 978-1-61208-367-4

- Code generation enables enhanced consistency since it serves as a repository of reuse. Also, each application developed by automatic code generation will have the same structure independent of the developer.
- Quality will be improved since the amount of manually written code –which is a source of quality variations-, is decreased.
- Productivity is increased since code generators can produce thousand lines of code very fast and these codes are correct by construction.

III. PROPOSED METHOD

For the systematic distribution of domain knowledge and best practices throughout the component development process, we propose automatic code generation as a plug-in to the model-based IDE (i.e., Rhapsody [6]) used for software component development in our projects. The reusable components given in Figure 1 are targeted for automation since the quality of these components affects all the projects they have been used.

The proposed automation process has two main phases, as given in Figure 2, namely, (i) ICD transcription, which is the process of transforming natural language message definitions to machine readable XML (Extensible Markup Language) format, and (ii) model-based code generation.

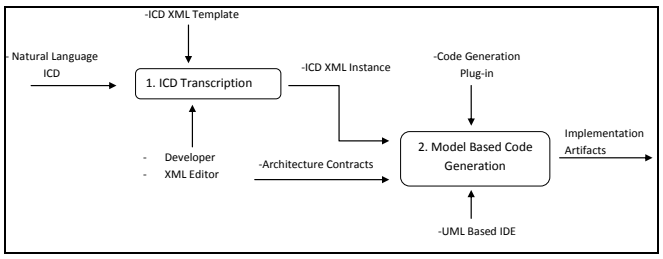


Figure 2. Proposed Automatic Code Generation Method.

One major obstacle on the automatic implementation of component-device communication protocol defined in ICDs is that the ICDs are prepared in natural language, and requires human interpretation. In order to accomplish automatic code generation, representation of the device ICDs in a machine readable format is mandatory. Another important problem is the variability of message structure defined in device ICDs. In the scope of this study, commonality & variability analysis is performed and device ICDs are modeled as an XML template, as given in Figure 3. The XML template given in Figure 3 will be described in detail in the following subsections.

A natural language ICD is the input to the ICD transcription phase where the proposed XML template given in Figure 3 is used as a guideline. The transcription activities are performed manually by the developer with the help of an XML editor. The output of this phase is an ICD XML instance, which is conformant with our domain model and includes the information content of a natural language document (such as communication parameters, message definitions, etc.) in a machine readable format.

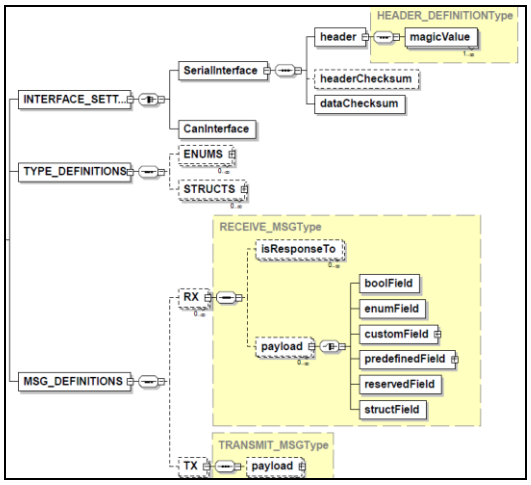


Figure 3. Proposed ICD XML Template.

A cross section of a sample XML instance, that models a "Sensor State" message with the message identifier "0x24" and contains payload fields for oil tank temperature and pump pressure is given in Figure 4. The oil tank temperature is expressed with 8 bit data which is mapped to a float variable in code generation phase. The mapping algorithm is described with the value of most significant bit (i.e. -100) and the precision value (i.e. 0.78125).

```
<RX msgId="0x24" payloadLength="8" msgLabel="SENSOR_STATE"
  <payload fieldName="oilTankTemp" order="1">
    <customField bitLength="8" targetType="float">
      <algorithm>
        <type2 MSBitValue="-100" Precision="0.78125"/>
      </algorithm>
    </customField>
  </payload>
  <payload fieldName="pumpPressure" order="2">
    <customField bitLength="8" targetType="float">
      <algorithm>
        <type2 MSBitValue="250" Precision="0.9765625"/>
      </algorithm>
```

Figure 4. A section from XML instance (Sensor State Message).

XML instance is the input to the code generation phase together with the architecture-based component contracts. Automatic code generation is performed by invoking the code generation rules embedded into a tool that is developed as a plug-in to our current development environment. At the end of code generation, the outputs are various implementation artifacts such as attributes, operations, events, type definitions, state chart diagram elements, etc. Some of the artifacts after code generation phase are given in Figure 5.

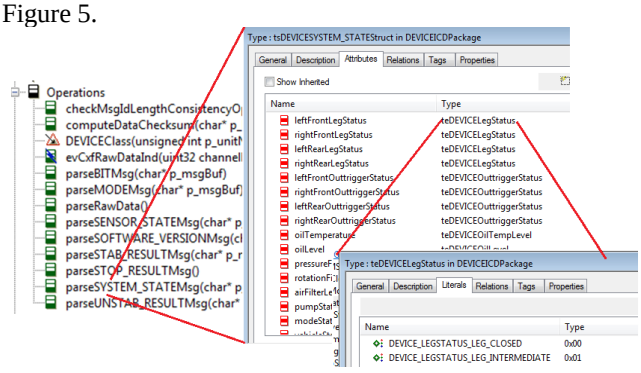


Figure 5. Snapshot of auto-generated model elements.

 ISBN: 978-1-61208-367-4

The detailed information on the construction of the XML template and the code generation activities are given in the following subsections.

A. ICD XML Template Structure

According to the model given in Figure 3, not every interface definition has a header checksum part, and in case of its existence the size and algorithm for its computation is variable. Also, data bytes in the payload part of messages can be converted to float or double values with some processing.

Although the structure of the messages defined in ICDs does not display variation depending on the direction of messages (transmit and receive directions), due to their semantics and behavioral differences our model distinguishes transmit and receive messages. From the components point of view, in addition to the content of a received message the information of to which transmit message it is a response to, is also important for the behavior.

Another variation point in message contents is how to decode the values encoded within byte sequences. Decoding methods extracted from the ICDs can be grouped as;

- Bit field definitions
- Float/double value transformation from discrete byte sequences with a given resolution
- Direct casting of byte sequences to short, integer, float, double, etc. values.

B. Implementation of Component Device Communication Protocol

For each component, there is a device (sensor, actuator, etc.) that it communicates over physical channels (serial, CAN, Ethernet, etc.) with conformance to an ICD. Since message parsing functionality is common for components in this study, this functionality is seen as the most suitable candidate for automation. Also, automatic implementation of enlisted messages, enumerated values, and numerical constants in the ICD are in the scope of this study.

TABLE I. XML TO IMPLEMENTATION MAPPING RULES

XML Element	Implementation Artifacts
Interface Settings	Serial port parameters (baud rate, parity, etc) and structure of messages (header, id, size, crc, etc.)
Type definitions and message payload	Enumerations and structures
Receive message names	Message specific parser function declarations
Receive message payload	Message specific parser function implementation
Decoding algorithm type	Converting byte sequences to target language types(float, int, etc.)
Checksum algorithm and data length	Checksum function implemention

Basic transformation rules can be applied after translation of ICD into XML file. The mapping rules from XML elements to implementation artifacts are described in Table I.

```
float tmpoilTankTemperature = (float)0;
unsigned short tmpoilTankTemperaturePlaceHolder = (unsigned short)0;
memcpy(&tmpoilTankTemperaturePlaceHolder, &p_msgBuf[0],sizeof(unsigned char));
tmpoilTankTemperaturePlaceHolder = cxfTtoH(tmpoilTankTemperaturePlaceHolder,interfaceEndian
tmpoilTankTemperature  = ((unsigned short)tmpoilTankTemperaturePlaceHolder*(float)0.78125);
if(tmpoilTankTemperature > (float)100)
{
    tmpoilTankTemperature = tmpoilTankTemperature - (float)200;
}
```

Figure 6. Code generated from XML instance.

Using the transformation rules and sample XML instance given in figure 4 automatic code generation is done. The automatically generated code for parsing oil temperature is given in Figure 6.

C. Implementation of Component Interfaces

Apart from the physical interface with devices, reusable software components also have contracts with the internal project specific components. While realizing these contracts different design alternatives can be adopted and different assumptions can be made by the developers. Also, it is difficult to prevent and diagnose the cases where different components have conflicting design decisions. In the scope of this study, critical interfaces and their expected design decisions are identified to provide a common behavior through the contracts. Our intention is to embed this common behavior into the component automatically by state chart design and reaction implementation.

TABLE II. CONTRACT TO IMPLEMENTATION MAPPING RULES

Contract Element	Implementation Artifacts
Component mode information	States
Component mode change indication	state transitions
Component activation request	state reception and response to activation request
Component setting request	State reception and response to component setting request
Component deactivation request	State reception and response to component deactivation request

Basic transformation rules from contract elements to implementation artifacts are given in Table II. Also a sample auto generated statechart implementation with the given rules is shown in Figure 7. The main states of the component, transitions between common states and common reactions are auto generated.

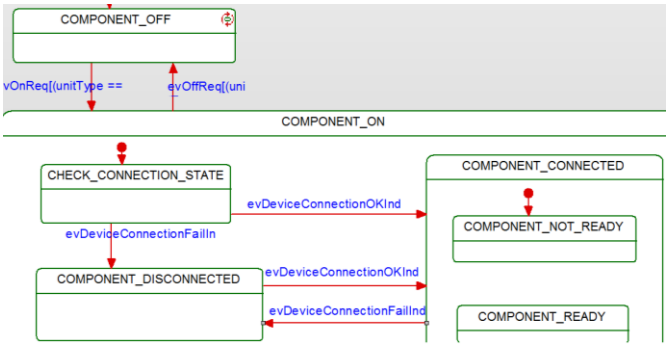


Figure 7. Example statechart implementation.

 ISBN: 978-1-61208-367-4

IV. CASE STUDY

In order to evaluate the effectiveness of the proposed method, a group of software components which are previously implemented with traditional methods are re-generated by the proposed method. Since these traditionally developed software components are already being used in the system, these components are seen as golden units. In order to provide the same functionality with golden units, automatically generated software components need to have approximately the same number of source lines. This assumption is based on the fact that automatic code generation process aims to generate the same code as traditional development. Under the given assumption, the effectiveness of proposed can be measured by comparing the source code line counts of automatically generated component and traditionally developed components.

TABLE III. CASE STUDY METRICS

Metric Name	SC1	SC2	SC3
Number of messages received	5	8	13
Number of data typed needed	4	12	21
Number of data fields transmitted	34	27	96
Total code line count for traditionally developed component	3524	3663	6378
Total code line count with proposed approach	1339	1357	2071
Auto-generated code rate(%)	37.9	37.04	32.42

The proposed automation method is applied to 3 different software components (SC1, SC2, and SC3), which are already available in our component repository. The source line count measurements related to traditionally and automatically developed components are given in Table III. In order to give information about the size and complexity of the interfaces, the number of messages in the receive direction, the number of distinct data types required to implement the content of the receive messages, and the number of data fields carried within receive messages are also shown in Table III.

The results indicate that proposed method is applicable to devices that have different interface complexities, since the automatically generated code ratio remains approximately the same for different components with different interfaces.

V. CONCLUSION AND FUTURE WORK

Based on our case studies, we can state that with the current scope of our proposed method, it is possible to achieve over 32% automatic code generation. It is estimated that this rate can be increased up to 50% with the addition of potential functionalities and behaviors that are scoped out in the first phase of our study. Considering that the automated code section handles most of the low level parsing operations and establishes a basis for the infrastructure of the components, we assess 30% as an effective automation rate for our domain. Another advantage of the automation is that it removed some mechanical actions during manual development, such as several type and function definitions and implementing predetermined reactions to requests in known states.

By enabling automatic code generation in one of the most error prone sections of component development, namely the "parser codes", we estimate that quality costs will be decreased in the long run and unit integration process can be completed more efficiently.

In addition to its direct effects to the component development, XML template based approach establishes a guideline for developers while inspecting the ICDs provided to them since it makes explicit the information content required for the accurate implementation of a component-unit interface. Although in scope of this study, the transformation of ICDs written in natural language to XML format is performed manually in order to increase the efficiency and usability of the proposed approach we plan to develop a wizard to guide the user during the ICD XML instance creation process. In the long run, we hope that software developers will not need to transcript ICD XML from the natural language document, but instead unit vendors will design their communication protocol on this wizard, thus its output will be ready to use by the code generation tools.

The current scope of the proposed method includes the message parsers, common states, transitions between common states, default reactions in the common states and the required attributes, types, and events to implement them. In the later phases of the study, the code generation capability will be extended to include the message sending functions and the realization of unit type specific interfaces.

REFERENCES

[1] R. Slaghi, and A. Strohmeier, "Better Generative Programming with Generic Aspects," Technical Report, Software Engineering Laboratory, Swiss Federal Institue of Technology, Switzerland, 2003.

[2] K. Czarnecki, "Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models," Ph.D. dissertation, Department of Computer Science and Automation, Technical University of Ilmenau, Germany, Oct. 1998.

[3] D. P. Gluch, A. J. Kornecki, and I. N. Sneddon, "Automated Code Generation for Safety-Related Applications: A Case Study," Proc. International Multiconference on Computer Science and Information Technology, pp. 383–391, 2006.

[4] K. Fertalj and M. Brcic, "A Source Code Generator Based on UML Specification", International Journal of Computers and Communications, Issue 1, vol. 2, 2008.

[5] S. Cullum, "The Effect of Automatic Code Generation on Developer Job Satisfaction," Technical Report No:2007/19, Open University, U.K., Sept. 2007

[6] E. Gery, D. Harel, and E. Palachi, "A Complete Life-Cycle Model-Based Development System," In Proceedings of the Third International Conference on Integrated Formal Methods, 2002, pp. 1-10.

 ISBN: 978-1-61208-367-4

Automatic Classification of Domain Constraints for Rich Client Development

Manuel Quintela-Pumares, Daniel Fernández-Lanvin, Alberto-Manuel Fernández-Álvarez, Raúl Izquierdo
Computer Sciences Department
University of Oviedo
Oviedo, Spain
e-mail: manuel.quintela@gmail.com, dflanvin@uniovi.es, alb@uniovi.es, raul@uniovi.es

***Abstract*— The current trend in web development, powered by the popularization of technologies like Ajax or platforms like iOS and Android, leads developers to gradually leave the classic *light-weight* web client in favor of *rich clients.* These clients manage not only presentation logic, but also business rules that transform part of the domain model that afterwards must be reintegrated in the server. This temporary duplication and transformation of part of the domain model force developers to deal with the management of the domain constraints that must be retrieved and applied in the client. This is a complicated and error prone task that usually involves redundant design and implementation on both sides. This work describes a tool that, given a domain model with its complete set of constraints, and the subset of classes that are required in the client, automatically identifies those constraints that the client requires and that can be applied separately from the server, classifying them according to their level of dependency with the server.**

Keywords-rich clients; constraints; OCL; UML.

I. INTRODUCTION

The architecture of web applications has been continuously evolving since the popularization of the primitive transactional script based systems. The current trend, powered by the popularization of technologies like Ajax or mobile platforms where native applications connect to the internet, leads developers to gradually leave the classic web *light-weight* client model, in which the client deals mostly with presentation logic [1], to a more distributed model, in which a Subset of the Domain Model (SDM) is retrieved and transformed in the client, to be redelivered back to the server to be reintegrated with the complete domain model (CDM) located on the server [1][2][3]. Some well-known web applications, like Google Docs or Google Calendar, are good examples of this approach.

This rich Internet application (RIA) architectural model carries a better user experience, since the classical delay between requests is mitigated [4]. However, it also involves important issues during the design and implementation [5]. The temporary splitting of the domain model, and its later reintegration in a multi user environment, force developers to figure out which of the constraints of the model should be checked in the client [6], whether they should be transformed, and which and how they should be checked again once the transformed sub-domain is reintegrated in the server [7].

Identifying at design time the constraints that can be safely verified on the client is a tricky job, and finding out if the existing ones can be modified -so that they can be located on the client- is a complicated and error prone task. Even when some constraints could be fully checked on the client, a redundant checking must be done back in the server for security reasons [8], requiring a redundant implementation.

Also, if there are different teams working at client and server side, human coordination problems can lead to inconsistencies. This problem is aggravated by the ever present changes in the requirements, making the constraints variable in both client and server. All these elements make the design and implementation of constraints a very complex, tedious and error prone task, especially as requirement changes accumulate over time [8].

All these problems would be avoided if we could automatically determine which of the constraints can be checked in the client and which cannot, and how they should be managed all along the process. This would support de dynamic generation of the control logic that manages those constraints in the client, avoiding redundant implementation and turning the development process more agile.

In our understanding, all the information we need for that can be deduced, for a specific SDM, from the information contained in the CDM in terms of entities, relationships and constraints.

To address these problems, we have designed a tool that can aid developers to easily produce the client subset using the CDM, its UML (Unified Modeling Language) [9] class diagrams and OCL (Object Constraint Language) [10] constraints as input parameters. A new class model will be generated for the client, maintaining the relations according to that subset, and discarding all unrelated classes, relations, methods and constraints. Since some of the constraints will require information from the server to be checked, and involve different levels of coupling, the tool automatically identifies and classifies the constraints that are relevant to the client by their dependency degree: (a) Completely independent of the server, (b) Can be dependent to the server in some circumstances and (c) Completely dependent to the server.

The rest of this paper is organized as follows. Section II describes the method we propose for the automatic classification of constraints. Section III provides an example illustrating how the tool works. Section IV addresses the related work. Section V presents the conclusion and future work. The acknowledgement closes the article.

 ISBN: 978-1-61208-367-4

II. METHOD FOR THE AUTOMATIC CLASSIFICATION OF CONSTRAINTS

We propose a method where the designer creates the UML model for the CDM located on the server, including its constraints described in OCL, as he/she would usually do, and then determine the classes and interfaces from the server model that corresponds to the SDM. With this information, a new class model and a new set of constraints are generated for the client. The constraints for the client are analyzed and automatically classified according to their level of dependency with the server, detecting those that may be problematic and require special attention.

The tool we have developed for implementing this method is a programmatic API written in Java that automatically generates the SDM from the CDM. Its implementation is based on EMF Ecore [11] class models and OCL files [10]. The input is the Ecore and OCL files that describe the server model, and the classes that belong to the SDM. The output will be a new Ecore file with the class model for the client, a new OCL file with the constraints that can be checked on the client, and an additional text file with information about the modifications of the class diagram, and the analysis, classification and documentation about the constraints.

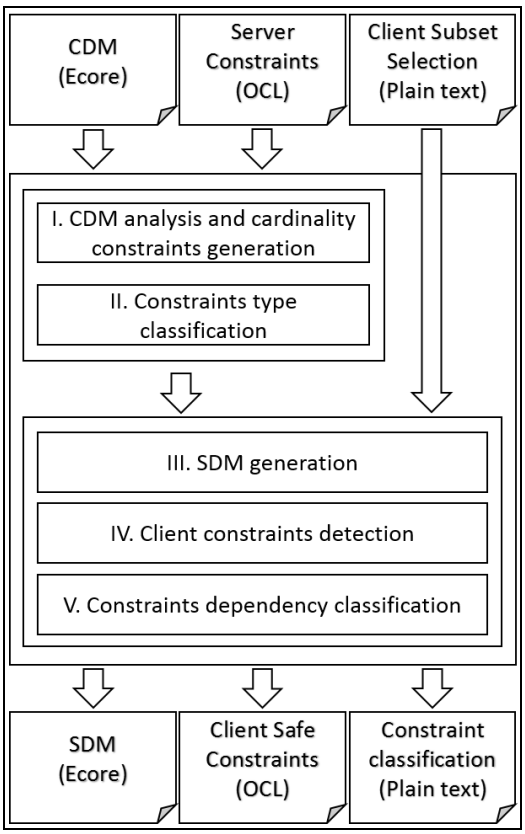


Figure 1. Inputs, outputs and processes that the tool carries out.

A. Analizyng the CDM and its constraints

The tool first analyzes the classes in the model, their attributes, methods and relationships. To ease the analysis of the cardinality constraints described in the class model relationships, those constraints are automatically transformed to OCL language, so that they can be processed homogenously with the rest (Figure 1, I.).

For every constraint, it collects information about the classes that are being referenced in its body, as well as the attributes that are being referenced and their primitive types, or the return type and parameters that are being used from their methods.

With all this information, the constraints are classified (Figure 1, II.) applying the following criteria:

- *Attribute constraint*: A constraint that only concerns a single attribute of the context class. We deduct this by observing the parameters that receive the operations of the constraint. If it contains a single property call whose type is of a primitive type, it is classified as an attribute constraint.
- *Object constraint*: A constraint that concerns more than one attribute of the context class. We deduct this as we did with the attribute constraint. If it contains different property references whose types are primitive types, it is an object constraint.
- *Class constraint*: A constraint that concerns several instances of the context class, and not elements of any other class. We determine this by observing if the types of the references (navigations, property accesses or method invocations) or parameter calls correspond to the context class, and not any other classes.
- *Domain constraint*: A constraint that makes reference in its operation to elements of other classes different than the class of its context. We calculate this in the same way that class constraints, but if a class has a different type than the context class, it is a domain constraint.

B. Generating the SDM for the client

After analyzing the CDM, the tool uses the subset of classes that the designer has selected to generate the SDM (Figure 1, III.). The new class model will contain only the classes described in the client subset. The relationships affecting the SDM classes are maintained in the new model. Those that connect any of those SDM classes to any class outside the SDM are processed as follows:

- Association, aggregation and uses relationships: If a class within the client subset has any of these types of relationship with a class outside the client subset, the relationships and the classes outside the client subset will be removed from the SDM.
- Inheritance relationships: A parent class can exist without its child classes, but in a class model a child class does not make sense without its parent classes. To address this problem, if a child class is included in the client subset by the designer, the tool automatically

 ISBN: 978-1-61208-367-4

includes its parent class. If there are various levels of inheritance above the selected child class, all the inheritance hierarchy for that class will be recursively included in the client subset.

- Interface relationships: We take the same approach as in inheritance relationships.
- Composition relationships: Composition is a relationship that models a strong relationship between a component and a container class, tying their lifecycles tightly. We consider that a component class can make sense without its relationship with a container class, but not the other way around. If the client subset includes a container class, we automatically include also its component classes and their composition relationships even if the designer did not consider them for the client subset. As in inheritance relationships, if the automatically included component classes are also containers of other classes, their components will be recursively included in the client subset.
- Methods: If the classes included in the client subset contain methods whose signature contains classes outside the client subset, those methods will be deleted from the class. We consider that, if those classes are kept outside the client subset, the methods that make reference to them will not be needed on the client.

C. *Selecting and classifying the constraints for the client*

The tool will select the OCL constraints whose context matches the elements in the client subset. The rest of the constraints will be discarded for the client (Figure 1, IV.). Constraints whose context is not in the SDM will not be considered due to the fact that there will not be any object of those classes in the SDM object graph.

The tool will also warn the designer about the level of dependency of each constraint with the server (Figure 1, V.). We define three levels of dependency:

1. *Completely independent*: All *attribute* and *object* constraints are completely safe for being checked independently on the client, since all the elements needed to check those constraints are already within the SDM object graph.
2. *Potentially dependent*:
   a. *Class* constraints may or may not be checked safely within the client. This will depend upon how the behavior of the client objects is defined. If every object of that class is always on the client, the constraint will be always safe. If the objects are requested from the server under request, the constraint could not be safe without some previous communication with the server in order to retrieve the required objects.
   b. *Domain* constraints that exclusively make reference to classes within the client subset are in the same circumstances as the *class* constraints. Their safety depends on the way the model is being managed. If a constraint needs information from objects that are not currently on the client, communication with the server will be required.
3. *Completely dependent*: *Domain* constraints that make reference to classes that are not in the client subset will always be dependent from the server, since they reference elements that are not considered on the client. These constraints should be delegated to the server, or when possible, be reformulated by the designer so that at least part of their operations can be checked on the client, delegating the rest to the server.

The output of this whole process is an Ecore file with the resulting SDM, a text file with the results of the modifications made from the CDM, and the analysis of the constraints related to the client and their classification. It also generates an OCL file containing the constraints that can be checked on the client without modification (those classified as *completely independent* or *potentially dependent*), and excluding the *completely dependent* (they cannot be checked on the client without modification).

III. THE ROYAL AND LOYAL EXAMPLE

The Royal and Loyal model [10] is a popular example usually used to explain the OCL language. We used a version of it to show the way the tool works if we need to develop a rich client for managing the addition of new Loyalty Programs. Figure 2 shows the Ecore model of the CDM located on the server, simplified for displaying only class names and references.

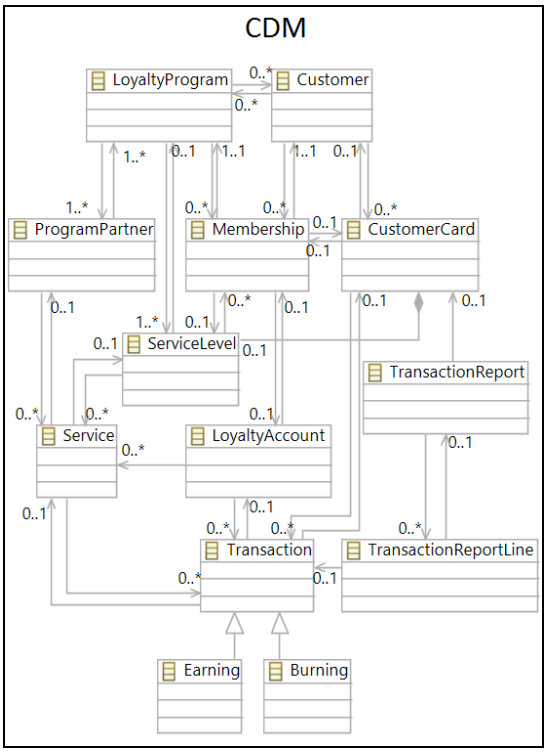


Figure 2. The Royal and Loyal Ecore model as CDM.

The version we used for this example has 22 constraints defined. After being analyzed by the tool, it automatically generates 28 additional constraints based on the cardinalities of the relationships of the class model, resulting a total of 50 constraints to process.

The model in our rich client will have the following classes from the CDM located on the server: "*Service*", "*ServiceLevel*", "*LoyaltyProgram*" and "*ProgramPartner*". Those classes will allow us to define new Loyalty Programs, partners, and the services they provide. The other functionalities that the full model provides, such as defining customers or managing their subscriptions to loyalty programs are out of the scope of this client.

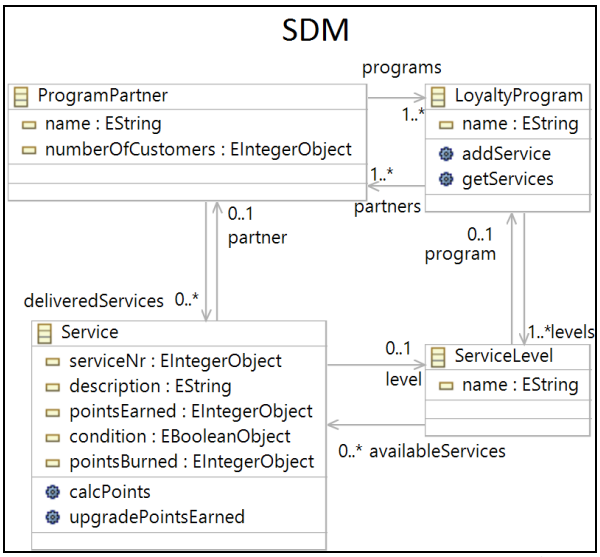


Figure 3. The resulting Ecore model as SDM generated by our tool for the client.

In Figure 3, we show the resulting SDM. Figure 4 presents the information it provides about the methods that have been deleted from the original class (due to their dependence from elements outside the client model), and also about the relationships that have been deleted from the original model.

```
Deleted classes:
Transaction, Customer, CustomerCard, Membership,
LoyaltyAccount, Burning, Earning, Transaction Report,
TransactionReportLine
-----------------------------------------------
Deleted Methods:
Customer-> enroll, selectPopularPartners,
enrollAndCreateCustomer, addTransaction, getServices
-----------------------------------------------
Deleted relationships:
Service ->Transaction: transactions
LoyaltyProgram -> Membership: memberships
LoyaltyProgram -> Customer: participants
ServiceLevel -> Membership: membership
```

Figure 4. The tool generates information about the classes, methods and relationships that are deleted in the process.

Regarding the constraints, it generates a plain text file describing those that affect each class, classifies them, and points out if they can be checked on the client or not. It detects 14 related to this SDM, 13 of them are classified as *domain* constraints and 1 as *attribute* constraint. After analyzing the dependency of these constraints, 1 is detected as *completely independent*, 9 as *potentially dependent*, and 4 as *completely dependent*. Figure 5 shows one constraint of each level of dependency as an example.

```
context Service::upgradePointsEarned(amount : Integer)
post postServiceUpgradePointsEarned: calcPoints() =
calcPoints@pre() + amount
        Classification: attribute
        Context Class: Service
        Referenced Classes: []
        Classes in context operation: [Service]
        Dependency: Completely independent
-----------------------------------------------
context LoyaltyProgram inv firstLevel:
levels->first().name = 'Silver'
        Classification: domain
        Context Class: LoyaltyProgram
        Referenced Classes: [ServiceLevel]
        Dependency: Potentially dependent
-----------------------------------------------
context ProgramPartner inv totalPoints:
deliveredServices.transactions.points->sum() < 10000
        Classification: domain
        Context Class: ProgramPartner
        Referenced Classes: [Service, Transaction]
        Dependency: Completely dependent
```

Figure 5. A selection of three of the resulting constraints, each one with a different level of dependency.

There are some constraints that can always be checked on the client without communicating with the server, like the postcondition for "*upgradePointsEarned*".

Some of the constraints have all the elements needed for checking the constraint in the client model, but it may need to communicate with the server to update the data, like the "*firstLevel*" invariant.

Other constraints reference elements outside the client model, that is, objects of that class don't exist on the client, like the "*totalPoints*" invariant.

The problem of having constraints on the client that reference elements that only exists on the server can be solved in several ways. The most straightforward way would be delegating the checking to the server. However, if we still want to make the checking on the client, it can be achieved by adding some kind of proxy that requests from the server the dependent values needed to check that constraint.

Finally, it creates an OCL file with the constraints that are *completely independent*, and *potentially dependent* (10 in total). It excludes the *completely dependent* ones since they refer to classes that are not on the SDM. The user should use this information to figure out the best way to adapt those dependent constraints for the SDM.

 ISBN: 978-1-61208-367-4

IV. RELATED WORK

There are several proposals that encourage locating more responsibilities on the client side rather than delegating them to the server. Hallé and Villemaire's [12] proposal is centered on rich clients that connect with web services, a system that checks the preconditions defined on the service interface on the client side before making the request, avoiding an unnecessary expenditure of resources on the server. Heidegger and Thiemann [13] add annotation-like pre and postcondition support to Javascript, a language widely used to develop rich clients where developing complex business logic is more common every day. The work presented by Zhang [14] suggests to move all business logic to the client, and leaving on the server only a database accessible through REST services. Leff and Rayfield [15] show a client designed to work in mobile environments where connection can be lost, defining mechanisms that support offline functionality and maintaining the integrity when the client is back online. All these proposals recognize the benefits of moving tasks to the client side and try to address some aspects of making integrity checks on clients. However, in all of them the responsibility of deciding which constraints are relevant on the client must be manually done by the developer.

Other authors try to solve the implementation problems of having constraints on a rich client [16][17], since popular tools to address business rules and validations are still very limited on this scenario. Rule engines, like Drools [18] or ILog [19], are a suitable solution for the server side, but they are not designed to deal with the ones located on the client. Tools to address client side validations like Struts [20], jQuery Validation Plugin [21], or Simfatic [22] are still limited to simple form checking, but are not designed to cover the complexity that client side business rules can require.

Liang et al. [16] propose a system in which validations are defined on an XML file, managing constraints that involve a combination of several attributes on the client's forms. This automates the implementation of part of the client side constraints, and improves the maintenance process. However, they explicitly left out of their scope the more complex and problematic *class* and *domain*. Schmidt et al. [17] designed a rule engine for the client side based on the RETE algorithm, where the constraints are defined on a file on the server. While they support the definition of complex constraints and even their delegation of to the server, the specific constraints affecting the client have to be manually specified. Most of these solutions would benefit with our proposal.

Louwsma et al. [7] analyzes the problems derived from managing constraints in a rich client for a GIS, where the user can add elements to the map over a graphic interface that will be updated to a central database. They propose a framework based on UML and OCL for the specification of constraints, and suggest several constraint classification criteria, but their implementation is hard-coded and delegates all the constraint checking to the database. They identify the problem of having constraints that can affect both client and server, proposing as future work that some types of constraints should be validated on the client for a better user experience, as well as automatic classification and detection of conflicting constraints, and their automatic implementation from a central specification.

Other previous works specifically address the problem of deciding how to split applications between different machines in an automatic way. Proposals like J-Orchestra [23] or Coign [24] process existing compiled applications, analyzing the way their different elements communicate. By means of code instrumentalization, they provide stubs to allow the division in different parts that can communicate, maintaining the same functionality. Also, Yang et al. designed a platform based on the Hilda language [25] with a runtime in both client and server that decides dynamically which elements of the application should run on the client and which on the server, basing on the characteristics of the client device.

All these approaches use different strategies to decide which the optimal distribution of their components is, by gathering information about the application behavior (like communication delay between elements, the size of the data transmitted, memory usage, capacity of the devices, or the demand by users of a certain functionality). However, none of these proposals deals with the problem of constraint redistribution. They add proxies to communicate the different split elements of the original design but do not change them to support constraint checking in order to maximize UI usability and responsiveness. All these solutions could benefit from automatic constraint classification and modification techniques in those cases in which client responsiveness is a priority.

Outside the scope of rich client development, techniques for automatically adapting OCL constraints have been developed to fit different purposes. Hassam et al. [26] propose techniques for automatically maintaining the consistency of the OCL constraints after applying modifications to the UML model. For each change made to a model, their tool identifies the OCL constraints affected by it, and then decides if the constraints have to be removed because they are no longer relevant, of if they can be automatically modified to be consistent with the modified model. Cabot and Teniente [27] developed techniques for automatically modifying constraints and domain models to achieve a more efficient integrity checking. For doing that, they develop techniques for simplifying OCL constraints, identify which operations trigger certain OCL constraints, and reformulate the constraints in the most efficient way given the possible operations found in the model.

These proposals acknowledge the problem of delegating to the designer the task of revising existing OCL constraints for achieving certain objectives when that tasks can be deduced from the UML model. In addition to this, although they are designed to solve scenarios different than the one we

 ISBN: 978-1-61208-367-4

propose, the principles behind the identification of which OCL constraints need attention, and some of the automatic modification mechanisms described in them could be useful for future developments of our tool.

V. CONCLUSION AND FUTURE WORK

The proposed tool deals with the generation of the new domain model for the client, selecting and classifying the constraints for the client, and automatically identifying the conflicting elements of the constraints that are not completely independent from the server. At its current state, it removes from the designer the responsibility of modeling the part of the client class model that overlaps with the server, providing useful documentation about the constraints that potentially affects the client.

If the designer wants to make a domain model on the client where as many as possible validations are made locally, the tool can help him/her to make better informed decisions while trying to modify the constraints and the client model to fit that purpose.

This approach can also complement the existing tools that deal with the implementation of constraints on the client, but currently delegate to developers the responsibility of organizing them.

We have previously developed means to achieve automatic error recovery in rich clients [28], letting the developer to choose which parts of the model require this mechanism and which do not, so that the overhead this recovery techniques involve is avoided where not needed. We believe that the information this tool provides can be used to find a way to automatically identify the parts of the client model that may benefit from the automatic error recovery and discard the ones that do not.

These tasks of analyzing, identifying and classifying the constraints managed with this tool are a first step. With this support, we can use this information to automatically modify the domain model and its constraints in a way that the resulting client can validate as many constraints as possible, minimizing communication with the server, and relieving the designer from finding out the required transformations that can be deduced automatically. Techniques for the automatic modification of constraints and domain models to achieve a more efficient integrity checking have already been studied, like the ones proposed by Cabot and Teniente [27], as well as techniques for adapting OCL constraints after the modification of UML models like Hassam et al. [26] proposals. We believe we can adapt some aspects of these techniques for our future needs regarding the automatic modification of constraints.

ACKNOWLEDGMENT

This work has been funded by the Department of Science and Technology (Spain) under the National Program for Research, Development and Innovation: project TIN2011-25978 entitled Obtaining Adaptable, Robust and Efficient Software by including Structural Reflection to Statically Typed Programming Languages.

REFERENCES

[1] J. Duhl, “White paper: Rich internet applications,” Tech. report, IDC, 2003.

[2] J. Allaire, “Macromedia Flash MX—A next-generation rich client,” Macromedia white Pap., no. March, 2002.

[3] J. Garrett, “Ajax: A New Approach to Web Applications | Adaptive Path,” 2005. [Online]. Available from: http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/ 2014.08.20

[4] A. Bozzon, S. Comai, P. Fraternali, and G. Carughi, “Conceptual modeling and code generation for rich internet applications.,” in Proceedings of the 6th international conference on Web engineering ICWE 06, 2006, vol. 1, p. 353.

[5] J. Preciado and M. Linaje, “Designing rich internet applications with web engineering methodologies,” Web Site Evol., pp. 23–30, 2007.

[6] A. Mesbah and A. Van Deursen, “An Architectural Style for Ajax,” in 2007 Working IEEEIFIP Conference on Software Architecture WICSA07, 2006, pp. 9–9.

[7] J. Louwsma, S. Zlatanova, R. Lammeren, and P. Oosterom, “Specifying and Implementing Constraints in GIS—with Examples from a Geo-Virtual Reality System,” Geoinformatica, vol. 10, no. 4, pp. 531–550, Jan. 2007.

[8] Z. L. Z. Liang and S. J. S. Jianling, “A field-oriented approach to web form validation for Database-Isolated Rule,” in 2009 IEEE International Conference on Systems Man and Cybernetics, 2009, no. October, pp. 4607–4612.

[9] “Object Management Group: UML 2.4.1 Superstructure Specification.” [Online]. Available from: http://www.omg.org/spec/UML/2.4.1/ 2014.08.20

[10] J. Warmer and A. Kleppe, The OCL, Second edition. Addison-Wesley, 2003.

[11] “Eclipse Modeling Framework Project - EMF.” [Online]. Available from: http://www.eclipse.org/modeling/emf/ 2014.08.20

[12] S. Hallé and R. Villemaire, “Browser-based enforcement of interface contracts in web applications with BeepBeep,” Comput. Aided Verif., pp. 648–653, 2009.

[13] P. Heidegger and P. Thiemann, “JSConTest: Contract-Driven Testing and Path Effect Inference for JavaScript.,” J. Object Technol., vol. 11, no. 1, p. 6:1, 2012.

[14] W. Z. W. Zhang, “2-Tier Cloud Architecture with maximized RIA and SimpleDB via minimized REST,” Comput. Eng. Technol. ICCET 2010 2nd Int. Conf., vol. 6, pp. V6–52–V6–56, 2010.

[15] A. Leff and J. Rayfield, “Programming model alternatives for disconnected business applications,” Internet Comput. IEEE, no. June, pp. 50–57, 2006.

[16] Z. L. Z. Liang and S. J. S. Jianling, “A field-oriented approach to web form validation for Database-Isolated Rule,” in 2009 IEEE International Conference on Systems Man and Cybernetics, 2009, no. October, pp. 4607–4612.

[17] K. Schmidt, R. Stühmer, and L. Stojanovic, “Gaining reactivity for rich internet applications by introducing client-side complex event processing and declarative rules,” in AAAI 2009 Spring Symposium: Intelligent Event Processing, 2009, pp. 67–72.

[18] “Drools - JBoss.” [Online]. Available from: http://drools.jboss.org/ 2014.08.20

[19] “IBM - ILOG,” Mar-2014. [Online]. Available from: http://www.ibm.com/software/info/ilog/ 2014.08.20

 ISBN: 978-1-61208-367-4

[20] "Apache Struts 2 Validation." [Online]. Available from: http://struts.apache.org/development/2.x/docs/validation.html 2014.08.20

[21] "jQuery Validation Plugin." [Online]. Available from: http://jqueryvalidation.org/ 2014.08.20

[22] "Simfatic Forms." [Online]. Available from: http://www.simfatic.com/ 2014.08.20

[23] E. Tilevich and Y. Smaragdakis, "J-orchestra: Automatic java application partitioning," in ECOOP '02 Proceedings of the 16th European Conference on Object-Oriented Programming, 2002, pp. 178–204.

[24] G. Hunt and M. Scott, "The Coign automatic distributed partitioning system," in OSDI '99 Proceedings of the third symposium on Operating systems design and implementation, 1999, no. February, pp. 187–200.

[25] F. Yang et al., "A unified platform for data driven web applications with automatic client-server partitioning," in Proceedings of the 16th international conference on World Wide Web - WWW '07, 2007, p. 341.

[26] K. Hassam, S. Sadou, V. Le Gloahec, and R. Fleurquin, "Assistance System for OCL Constraints Adaptation during Metamodel Evolution," in 2011 15th European Conference on Software Maintenance and Reengineering, 2011, pp. 151–160.

[27] J. Cabot and E. Teniente, "Incremental integrity checking of UML/OCL conceptual schemas," J. Syst. Softw., vol. 82, no. 9, pp. 1459–1478, Sep. 2009.

[28] M. Quintela-Pumares, D. Fernández-Lanvin, R. Izquierdo, and A.-M. Fernández-Álvarez, "Implementing automatic error recovery support for rich web clients," in WISE'10 Proceedings of the 11th international conference on Web information systems engineering, 2010, pp. 630–638.

 ISBN: 978-1-61208-367-4

A Classification Schema for Development Technologies

Davide Taibi, Christiane Plociennik
University of Kaiserslautern
Kaiserslautern, Germany
{taibi, christiane.plociennik}@cs.uni-kl.de

Laurent Dieudonné
Liebherr-Aerospace
Lindenberg, Germany
laurent.dieudonne@liebherr.com

***Abstract*— Software and hardware development organizations that consider the adoption of new methods, techniques, or tools often face several challenges, namely to: guarantee process quality, reproducibility, and standard compliance. They need to compare existing solutions on the market, and they need to select technologies that are most appropriate for each process phase, taking into account the specific context requirements. Unfortunately, this kind of information is usually not easily accessible; it is incomplete, scattered, and hard to compare. Our goal is to present a case study on a classification schema we applied on the avionic domain to help decision makers to easily find, compare and combine existing methods, techniques, and tools based on previous experience. The results show that the schema helps to transfer knowledge between projects, guaranteeing quality, reproducibility, and standard compliance.**

Keywords—process improvement; technology classification; technology selection; tool selection; method selection; process configuration.

I. INTRODUCTION

The software and hardware market is evolving continuously and companies that develop software or hardware need to keep improving their processes by introducing new technologies, in order to be able to keep pace with other competitors on the market.

Finding a product development process that guarantees quality and reproducibility often takes years. Moreover, in certain domains, such as avionics, the process must comply with a set of standards, such as DO-178 [13].

The introduction of a new technology may break the consistency and standards compliance of the process. To limit this risk, two major aspects must be considered. First, the objectives and prerequisites for each process step must be fully documented and structured. Second, the contribution of each method and tool intended to be used must be limited to the objectives set by each domain process activity and their role in each process step must be fully described.

A structured framework, enabling the classification of the technologies in the process activities would speed up the integration of new technologies and contribute to guaranteeing compliance with the company processes.

To facilitate the classification of technologies, the Reference Technology Platform (RTP) has been developed. RTP is a set and arrangement of methods, workflows, and tools that allow interaction and integration on various levels in order to enable efficient design and development of (complex) systems [20].

In the context of the ARAMiS project [16], a classification schema based on the RTP has been developed. It classifies technologies along two dimensions: abstraction levels and viewpoints.

In this paper, we present a use case on the application of this schema in the avionic domain. Moreover, we also introduce an implementation of the schema we developed: the Process Configuration Framework Tool (PCF) [8].

The results of this work suggest that the classification provides a useful framework for decision makers and allows them to base their decisions on previous experience instead of on personal opinions. Moreover, the classification allows them to guarantee process quality, reproducibility and standards compliance. Finally, it facilitates knowledge transfer from project to project or between employees.

The remainder of this paper is structured as follows: Section II describes related work; Section III introduces the classification schema, while Section IV describes the avionic use case. In Section V, we introduce the PCF tool and discuss the benefits of the schema in Section VI. Finally, we draw conclusions in Section VII and provide an outlook on future work.

II. RELATED WORK

In this section, we introduce the most common technology classification schemas.

An early work on technology classification is Firth et al. [19] from 1987. It classifies software development methods according to the *stages* of the development process (specification, design, and implementation) and the *view* (functional, structural, and behavioral). This schema is two-dimensional like our schema, and its *views* dimension is similar to our *viewpoints* dimension. However, the second dimension is rather different: Firth et al. focus on the process stages, while we map these onto the viewpoints dimension. Our second dimension is concerned with abstraction instead.

Another early work is the *Experience Factory*, published in the late 1980s [3] and updated in 1991 [4] and in 1994 [5]. Here, software development artifacts are described in so-called *experience packages* along with empirical evidence on how they have been used previously. The main goal of the Experience Factory is to provide a framework for software reuse to help software engineers make decisions based on company experience.

Compared to our work, the Experience Factory is a more general concept. In the Experience Factory, an object for reuse can be any software engineering artifact, including products, requirements documents etc. Furthermore, the Experience Factory does not provide a specific schema for

 ISBN: 978-1-61208-367-4

storing different technologies for reuse, and it does not include algorithms for searching or combining technologies. Another approach to technology classification developed in parallel to the later experience factory versions is the *C4 Software Technology Reference Guide* (C4 STR), a catalog containing more than 60 technologies.

Compared to our work, the C4 STR provides a huge number of technologies in its schema. Nonetheless, compared to our schema, the attributes it uses are not as detailed and there is neither a reference to context nor to the impact.

The C4 STR was later merged with the Experience Factory approaches by Birk [5]. In the late 1990s, this evolved into a new concept of *experience management*. Based on this work, more publications evolving this schema and extending the Experience Factory idea [12] appeared.

Ploskonos [18] developed a classification schema for software design projects. The goal is to facilitate the adaptation of generic process descriptions and methods to individual processes. Design projects are classified into one of four groups: *Usability*, *Capability*, *Extension*, and *Innovation*. Each group is associated with certain process characteristics that help the user set up the actual process. This approach is narrower than ours: It focuses on classifying processes according to the project type, omitting other characteristics such as project size or domain.

III. THE CLASSIFICATION SCHEMA

In this section, we introduce the classification schema applied, as foundation for our case study. The schema is aimed at providing a complete engineering tool chain for collecting and integrating technologies to support the activities of a structured development process.

The paper addresses the development of big, complex projects in the industry, which are spread out over several years and occupy many employees.

Industries usually work with requirements-based process models planning the different baselines in order to ensure the accomplishment of these baselines on time via different phases of realization. Each phase and each step of the processes usually produces artefacts used as inputs for the next phase(s) or step(s). These models are derived from, or include, the V-Model [23], which is traditionally used inside the iterations made for the accomplishment of each baseline. Additionally to the iterations, other concepts like definition of phases, definition of objectives, periodical assessments, definition of roles, forward and backward traceability, etc. are traditionally used in these development processes, and have widely inspired current agile methodologies, like SCRUM [23].

The schema presented in this paper represents a generic development model covering the industry development processes. The instances of the generic development model are naturally dependent on the industry development standards and on the company itself.

The information provided in this schema, enables decision makers to find the most appropriate technology based on their interaction and integration on various levels. This contributes to the efficient design and development of complex systems. Furthermore, the schema can give an overview of methods and tools used in past projects. Via the different planning phases, assessment meetings and accomplishment summaries inherent to the industry processes and performed periodically during each project development, the decisions made, the quality and special uses of the tools, methods and technologies, can be collected during the whole development life cycle of each project. This contributes to building a knowledge database, addressing both best practices and pitfalls, adapted to the company development processes. Hence, new projects do not have to start from scratch, but can benefit from previous experience. The same applies for new employees: The schema can help them to familiarize themselves quickly with the methods and tools available for each phase of the development process. Thus, the schema facilitates knowledge transfer inside a company.

The schema can be represented as a matrix with viewpoints as columns and abstraction levels as rows. The viewpoints of the classification are defined as "*Requirements*", "*Functional*", "*Logical*", and "*Technical*". These viewpoints can be mapped to the three phases of the development process where the requirements viewpoint coincides with the requirements capture phase, the functional and logical viewpoints are related to the design phase, and the technical viewpoint is related to the construction or implementation phase (see Figure 1).

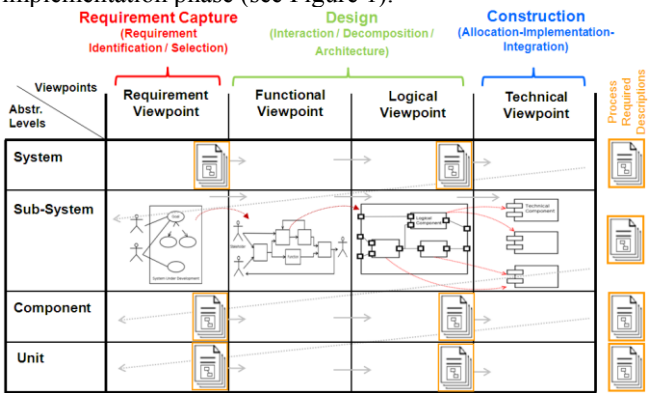


Figure 1. Generic representation of our classification schema.

In the generic version of the schema, the abstraction levels resemble the decomposition of the system into sub-systems, components, and sub-components or units (see Figure 1). For specific application domains (e.g., automotive, avionics and railways), a different, domain-specific set of abstraction levels can be defined. For example, in the avionics domain, abstraction levels are defined as *"Aircraft"*, *"System"*, *"Equipment"*, and *"Item"* (see Figure 2). Each cell of the schema represents a step of the product development process that must be performed starting from the topmost and leftmost cell to the rightmost, as shown by the arrows in Figure 1.

The output of each step leads to the realization of artifacts contributing directly to the fulfillment of the process objectives required by the domain or indirectly by focusing artifacts needed by other cells, which are inputs for later

 ISBN: 978-1-61208-367-4

steps. The objectives specified by the domain process depend on the development phase and the abstraction level.

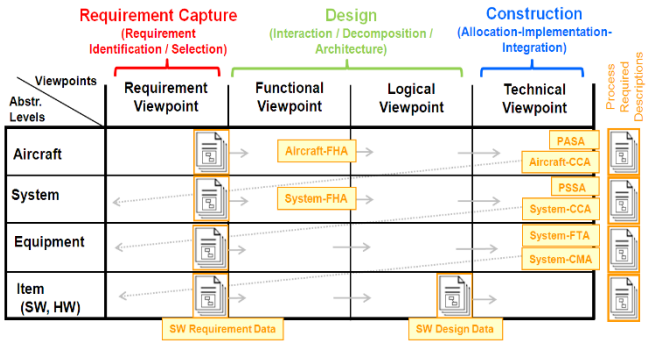


Figure 2. Example of classification schema for the avionics domain.

Starting at a given abstraction level, the requirements suitable for this abstraction level must first be captured in the requirement viewpoint. These filtered requirements are the outputs of this viewpoint and are necessary to start the design of the system. During the design phase, the function network determined in the functional viewpoint is needed first in order to perform decomposition and/or structuration of the identified functions, realized in the logical viewpoint. Once the objectives of the logical viewpoint have been achieved, the construction of the system can be started in the technical viewpoint. Iterations are possible, among others to introduce new requirements or to consider realization constraints appearing a posteriori that influence the system design.

At the end of an abstraction level, the requirements derived from the design and thus from the requirement viewpoint not being fulfilled at this abstraction level are used as a basis for the next abstraction level. They are captured in the requirement viewpoint at the new current abstraction level, where similar work as for the previous abstraction level is performed again.

To allow partial and iterative development, the transition from one cell to the next is controlled by a set of transition criteria. Transition criteria support the evaluation of the risks of starting the next development step although the objectives of the current step are only partially fulfilled. The current fulfillment of the objectives can then be controlled and will be realized after several iterations.

To fulfill the objectives of each matrix cell, the system and software engineers have to use methods that must mostly be supported by tools. Depending on the category of product to be developed, its requirements, the abstraction level, and the focus set in the current development iteration (e.g., which objectives are addressed), the methods and tools may differ, and the technology chain used can also be integrated differently. The transition criteria between the process steps must be supported by the methods as well.

IV. APPLYING THE CLASSIFICATION SCHEMA IN THE AVIONICS DOMAIN

In this section, we sketch an example of a use case of the classification schema in the avionics domain.

In the avionic industry, two main processes are defined and address two different aspects corresponding to the two branches of the V-Model: the *Development Process* and the *Integral Process* [14]. The combination of both main processes defines abstraction levels (Aircraft, System, Equipment/Item, Software, Hardware, etc.) and specific processes for each of them. Iterations can be done inside an abstraction level, or inclosing them. The overall resulting applicable development process can be summarized like the following suite of development phases, where the previous ones are required by the next ones: *Aircraft Requirements Identification*, *Aircraft Function Development*, *Allocation of Aircraft Function to Systems, System Requirements Identification*, *Development of System Architecture*, *Allocation of System Requirements to Items*, *Item Requirements Identification*, *Item Design* (corresponds to Software and Hardware Development, both having specific processes), *Item Verification*, *System Verification*, and *Aircraft Verification*.

These different phases can be well mapped onto the generic development model, among others by instancing the abstraction levels and by specifying the objectives of the viewpoints for each abstraction level, according to the company and project needs.

For example, at the system level, the *System Requirements Identification* corresponds to the *Requirement Capture Viewpoint*, the *Development of System Architecture* is realized via the *Functional* and *Logical Viewpoints*, the *Allocation of System Requirements to Items* belongs to the *Technical Viewpoint*, where the decision is taken on which technology will be involved to realized the *Items* (*Item Design* corresponds to Software and Hardware development). The *Verification* phases are realized in the *Technical Viewpoint* of corresponding abstraction levels, where the integration activity is performed. For each phase, objectives concerning safety assessments, validation, verification, etc. are defined via the *Integral Process* and should be met in order to move to the next phase, or must be accomplished during a next iteration. The same logic applies when moving to the next abstraction level.

The same principles apply for all the other abstraction levels. This is also true for the Software and Hardware development, but with different steps inside the phases and different objectives, because they are defined by specific processes specified in the avionics standards DO-178C [13] and the DO-254 [21].

We consider the development of a safety-critical system – a Flight Control System (FCS). We give an example on how the regular avionic development process, according to the civilian aircraft and systems development process guidelines ARP4754A [14], can be mapped on the classification schema (see Figure 2).

Here, we briefly introduce how to use the classification schema efficiently by describing the most important development process steps and their artifacts.

 ISBN: 978-1-61208-367-4

Based on the high-level aircraft requirements and design decisions, the requirements on the FCS must first be captured, expressed, and validated precisely (*requirement viewpoint*). The artifacts for this step are the functional and non-functional requirements that contain the goals of the system (e.g., "control the three axes of the aircraft: pitch, yaw, and roll"), the operational requirements (e.g., operational modes), the safety requirements (e.g., which criticality for which surface/axis), the high-level performance requirements (e.g., aircraft response time following cockpit control requests), etc. The requirement capture can be facilitated with use-cases, such as SysML/UML, or with requirements tools using structured text.

Once captured, the requirements must be validated, which is a transition criterion for proceeding to the next step. Different activities and requirements types are analyzed using different technologies, according to the avionics standards.

Based on these requirements, the behavior of the system is then analyzed and a functional architecture in the form of a network of the essential functions covering the major system functionalities must be formulated (*functional viewpoint*). An example of a major functionality at the system abstraction level is the altitude control via the pitch axis, which is realized by the elevator surfaces. Essential functions are those realizing the functionality and having an external interface with other parts of the system, for example actuator control, acquiring of the surface position, synchronization with the other surfaces, etc. For example, block definition diagrams from the SysML and signal flow diagrams are well suitable to model the functions network.

Once the definition of these functions and their related requirements is completed, a Functional Hazard Assessment (FHA) must be performed [14]. The FHA produces safety requirements and design constraints for the next design step which are necessary to make decisions about the decomposition and structuration of the functions in order to realize a suitable system design. In this next step (*logical viewpoint*), these essential functions are structured, completed, and/or decomposed in order to shape the components to be realized on this abstraction level – here named "logical components". The logical architecture determination is also efficiently supported by the SysML/UML technologies, and the behavior can be well designed via control flow diagrams, state machines, etc. Simulation technologies can be used to validate the interactions and behavior between the logical components, once they are correctly formalized.

Based on these components and their inherited requirements (the logical components are derived from the functions of the *functional viewpoint*, which are themselves derived from the requirements of the *requirement viewpoint*), technical solutions suitable for this abstraction level are identified or existing technical solutions are chosen (*technical viewpoint*). These technical solutions are called "technical components" in this paper. The requirements expressed by the logical components drive the selection of the technical components.

Iterations inside an abstraction level are feasible for introducing new requirements, or for increasing the reusability rate by considering already existing technical components. As a consequence, the structuring (decomposition and composition) of the logical components may be performed in a different way. A configuration management system is mandatory for managing the different alternatives and versions.

At the end of the technical viewpoint, different validation activities (part of the transition criteria) must be accomplished, like a Preliminary System Safety Assessment (PSSA), a preliminary common cause analysis (CCA), etc. [14] in order to validate the decisions made in the functional, logical, and technical viewpoints.

If the already existent technical components fulfill exactly the requirements expressed by the logical components mapped onto them, the work is completed and the associated requirements are considered as fulfilled. This is an ideal case of reusability and will probably not arise very often at higher abstraction levels such the Aircraft and the System levels, but may arise at the Equipment or Item level.

The technical components that do not exist yet or that do not completely fulfill the requirements expressed by the logical components mapped onto them, and the logical components that are still too complex to be allocated to a particular technical solution are both inputs for the next abstraction level. They express requirements that have not been fulfilled at the current abstraction level and must be dealt with at the next one. Thus, the work on the next abstraction level can start.

The traceability, required by avionics processes at the different abstraction levels, is performed 1) between the viewpoints of the same abstraction level and 2) between the abstraction levels. For this second case, the traceability is performed between the technical and logical viewpoints of a given abstraction level and the requirement viewpoint of the next abstraction level.

For example: For 1), the technical components (*technical viewpoint*) are assigned to the logical components (l*ogical viewpoint*) that drove their selection. For 2), on abstraction level AL, each technical component not already realized and each logical component that cannot be mapped to a technical component must be addressed on abstraction level AL-1. They express requirements to be captured in the *requirement viewpoint* of AL-1. The requirements expressed at the *Requirement viewpoint* of AL-1 are then linked to the requirements expressed by the corresponding technical and logical components from the abstraction level AL.

The other abstraction levels follow the same logic for each step with methodology objectives, process objectives and artifacts, and similar activities that need to be carried out. All of them can be well mapped in the classification schema.

For example, at the Aircraft abstraction level, similar process activities as for the system level are realized, like an FHA, a Preliminary Aircraft Safety Assessment (PASA), and a CCA. For the equipment abstractions level, a Fault Tree Analysis (FTA) is required as well as a Common Mode Analysis (CMA), etc. For the software abstraction level, the

 ISBN: 978-1-61208-367-4

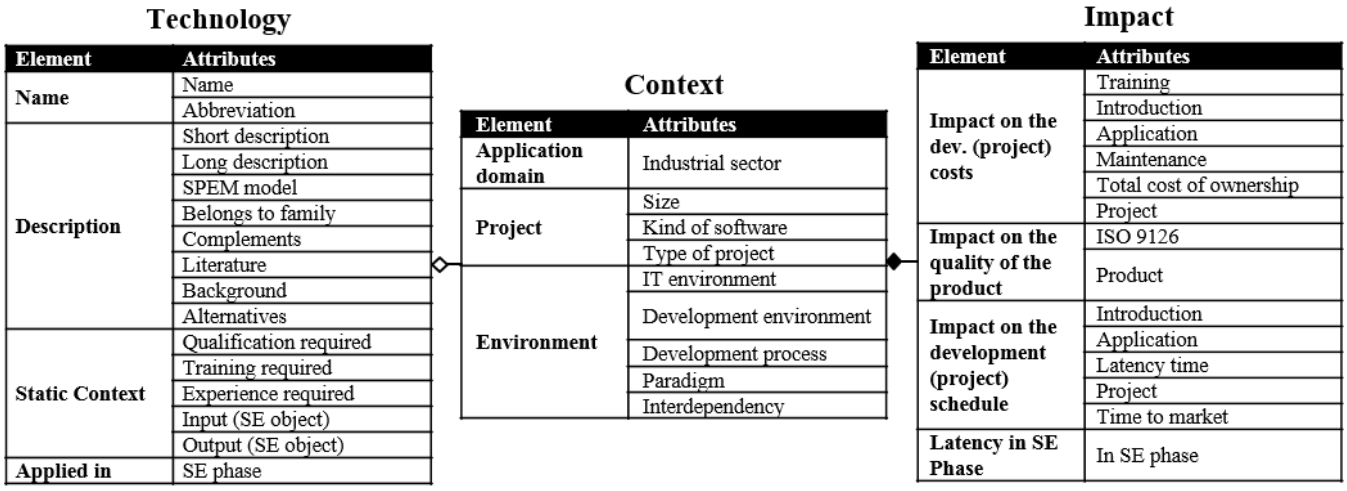


Figure 3. PCF Data Schema.

avionics standard DO-178 [13] defines different phases (called "processes", such as the Software Requirements Process and the Software Design Process) with several objectives requiring numerous artifacts, such as requirements and detailed design descriptions, validation and verification artifacts, etc., which can be performed by using different methods and tools (e.g., for verification: Classification Tree, Equivalence Partitioning, Cause-and-Effect Analysis), with each containing pros and cons, depending on the context of the current development.

Another issue that belongs to the top-down process explained here is that the reusability of existing solutions potentially fulfilling parts of the system also requires suitable and standardized methods and tools. Existing technical solutions may also consist of components developed outside the company, such as a microcontrollers, software libraries, etc. with other degrees of quality and using different processes. In any case, these existing solutions need to be completely and suitably characterized and must be integrated efficiently into the development process.

However, reusability is not a separate activity that can be transposed directly as a technology that can be integrated into the schema. In fact, it influences different activities, such as the decomposition in the design phase at the *logical viewpoint*, the accurate characterization of the existing solutions and the deployment activity at the *technical viewpoint*, etc. All these aspects related to reusability must also be taken into account in these activities. For example, it should be possible to integrate a systematic deployment process and its related techniques as explained by Hilbrich and Dieudonné [15] into the schema via these activities. As an example for this case, the software applications that are to be mapped optimally onto electronic execution units (ECU) need to be decomposed and structured in a way that makes them well compatible with the capabilities of the ECUs in order to allow the use of a minimum number of ECUs. However, on the other hand, the ECUs must be formalized completely and their description must be easily accessible by the system and software architects in order to influence the system design and to be correctly selected during deployment. In ARAMiS, we also provide a template for formalizing multicore processor capabilities in a form and on an abstraction level that can be used by system and equipment engineers. The formalization must be performed by the software and hardware engineers who design the ECUs. A noticeable advantage is to be able to validate per analysis or per simulation more aspects of the system, like the timing reactions, or the resource consumption.

These activities related to reusability are scattered across different cells of the matrix. At present, they need to be taken care of by the system designer. It would be helpful if they could be better integrated into the chain of methods and tools in the future.

V. IMPLEMENTING THE CLASSIFICATION SCHEMA IN PCF

The proposed schema has been implemented as a web application in the PCF tool [8]. PCF allows users to search for technologies based on abstraction levels and viewpoints as defined in the schema. Furthermore, PCF adds two more aspects to provide information about previous experience using a specific technology: *Context* and *Impact*. Hence, the data schema in PCF is based on three models as defined in [9] (as shown in Figure 3):

- *Technology*: includes a set of attributes for describing a technology in as much detail as possible.
- *Context*: includes information on the context, such as application domain, project characteristics, and environment in which the respective technology has been applied.
- *Impact:* includes previous experience on applying a specific technology in a specific context.

The PCF tool contains a search feature that allows users to search for technologies based on the attributes defined in the models in Figure 3. This enables the user to search for technologies used in projects with specific characteristics, e.g. projects fulfilling a certain industrial standard.

Basic use cases for PCF, as shown in Figure 4, are:

- Search for a technology based on context requirements (not mandatory)

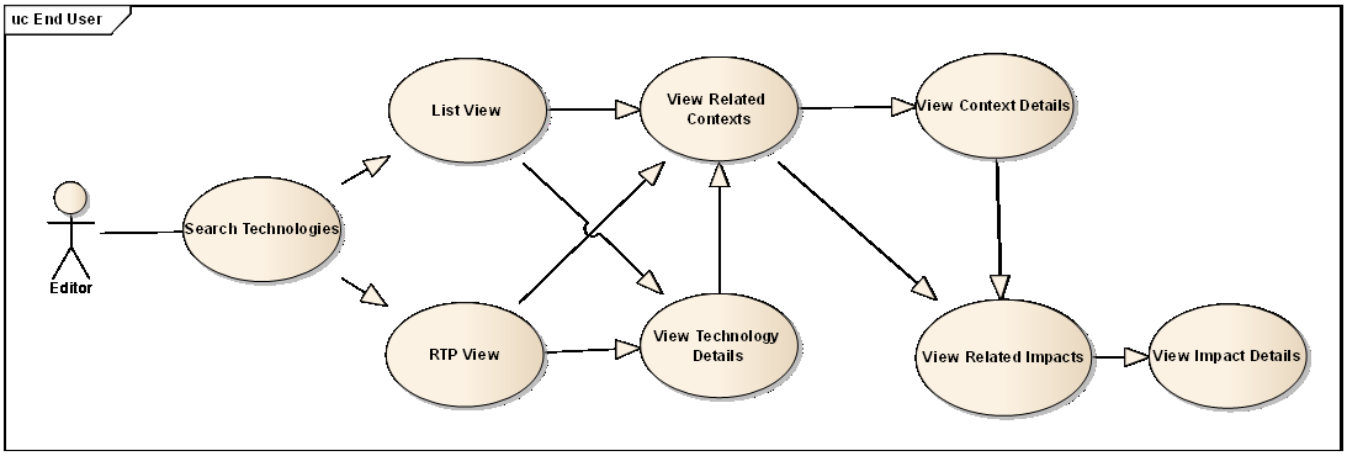


Figure 4. PCF Use Case.

- List view
- Matrix view

- View details for a technology
- View related context
- View details for a context
- View related impacts
- View details for a related impact

Moreover, PCF implements the schema for different domains (avionics, automotive, and railways).

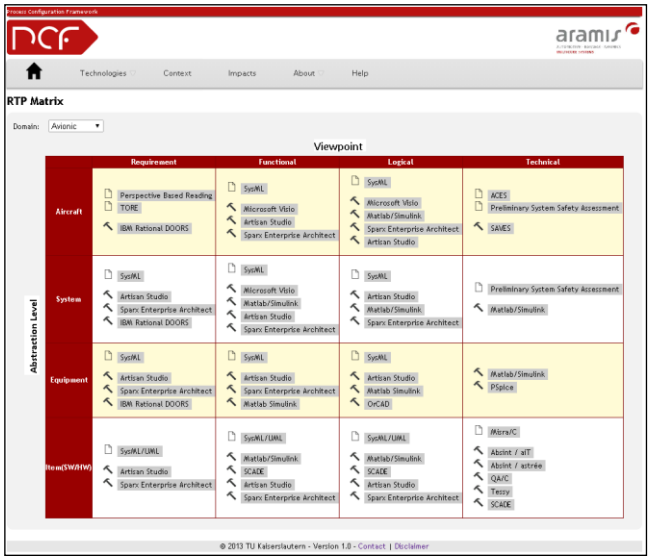


Figure 5. An example of the schema in the avionic domain implemented in PCF.

Figure 5 shows an example of the schema represented in PCF for the avionics domain. This figure includes the methods mentioned in the use case or directly the tools realizing them, as well as several other technologies for the avionics domain in addition to those mentioned above. In this version of the tool, we do not consider interoperability issues. The next version of the tool will address the challenge of interoperable tool chains.

VI. BENEFITS

The classification schema provides benefits for different people working in software projects, especially for project managers, software engineers, and technology providers (software and hardware vendors).

The use case indicates that, from the point of view of software engineers and decision makers, the classification schema provides an effective platform for searching for existing technologies. For industry domains strongly based on process based development, it also provides a toolbox for accurately specifying the use of each technology for rigorous process steps.

The main benefit for the ARAMiS project was that creating the classification schema for the avionics domain helped us to improve the schema. Several changes to the schema have been suggested based on issues raised during the application of the schema concept in practice. Another major benefit for the ARAMiS project was the identification and specification of methods and tools for improving the integration of multicore processors for safety-critical domains.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a use case reporting on the usage of a classification schema in the avionics domain and its implementation in the PCF tool.

The schema is aimed at collecting and integrating methods and technologies to support the activities of a structured development process. It allows decision makers to find the most appropriate technology based on their interaction and integration on various levels to enable efficient design and development of complex systems.

The schema provides a matrix representation of the development activities classified into viewpoints and abstraction levels that enables users to easily search for the most appropriate technologies throughout the whole development lifecycle.

The use case shows that the schema helps process managers to keep track of the technologies used in previous projects and to maintain traceability throughout the whole

 ISBN: 978-1-61208-367-4

process. Moreover, the schema can be useful to enable knowledge transfer inside the company.

Supported by the ARAMiS project and its partners, future work will include the collection of existing technologies to create a baseline for the platform. Moreover, we are planning to run an empirical study to validate the effectiveness of the schema.

ACKNOWLEDGMENT

This paper is based on research carried out in the ARAMiS project, funded by the German Ministry of Education and Research (BMBF O1IS11035Ü).

REFERENCES

[1] A. Rajan and T. Wahl, “CESAR - Cost-efficient methods and processes for safety-relevant embedded systems”, Springer, 2013, ISBN: 978-3-7091-1386-8.

[2] K. Pohl, H. Hönninger, R. Achatz, and M. Broy, “Model-based engineering of embedded systems - The SPES 2020 Methodology”, Springer, 2012, ISBN: 978-3-642-34614-9.

[3] V. Basili and D. Rombach, “Towards a comprehensive framework for reuse: A reuse-enabling software evolution environment”, Technical Report, University of Maryland, 1988.

[4] V. Basili, D. Rombach, “Support for comprehensive reuse”, Software Engineering Journal, vol. 6, Sep. 1991, pp. 303-316, ISSN: 0268-6961.

[5] V. Basili, G. Caldiera, and D. Rombach, “Experience factory”, In: Encyclopedia of Software Engineering, Marciniak, John J., Ed., New York: Wiley, pp. 469-476, 1994.

[6] A. Birk, “A knowledge management infrastructure for systematic improvement in software engineering”, PhD dissertation, Stuttgart, Fraunhofer IRB Verlag, 2000.

[7] K. Schneider, J.P. Hunnius, and V. Basili, “Experience in implementing a learning software organization”, IEEE Softw., vol. 19, May 2002, pp. 46-49.

[8] P. Diebold, L. Dieudonné, and D. Taibi, “Process configuration framework tool”, Euromicro Conference on Software Engineering and Advanced Applications 2014, in press.

[9] P. Diebold, “How to configure SE development processes context-specifically?”, 14th International Conference on Product-Focused Software Process Improvement (PROFES 2013), Springer, Jun. 2013, pp. 355-358, ISSN: 0302-9743.

[10] P. Diebold, C .Lampasona, and D. Taibi, “Moonlighting Scrum: An agile method for distributed teams with part-time developers working during non-overlapping hours”, Eighth International Conference on Software Engineering and Advances, IARIA, Oct. 2013, pp. 318-323, ISBN: 978-1-61208-304-9.

[11] A. Jedlitschka, N. Juristo, and D. Rombach, "Reporting experiments to satisfy professionals' information needs", Empirical Software Engineering, 2013, doi: 10.1007/s10664-013-9268-6. [Online]. Available from: http://publica.fraunhofer.de/documents/N-266529.html. Last access 2014.07.21.

[12] A. Jedlitschka, D. Hamann, T. Göhlert, and A. Schröder, “Adapting PROFES for use in an agile process: An industry experience report”, Sixth International Conference on Product-Focused Software Process Improvement (PROFES 2005), Springer, Jun. 2005, pp. 502-516, ISSN: 0302-9743, ISBN: 3-540-26200-8.

[13] RTCA DO-178C, “Software considerations in airborne systems and equipment certification”, Dec. 2011.

[14] SAE ARP4754 Rev. A, “Guidelines for development of civil aircraft and systems”, Dec. 2010. Available from: http://standards.sae.org/arp4754a. Last access 2014.07.21.

[15] R. Hilbrich and L. Dieudonné, “Deploying safety-critical applications on complex avionics hardware architectures”, Journal of Software Engineering and Applications (JSEA), vol. 6, May 2013, pp. 229-235, ISSN: 1945-3124.

[16] ARAMiS project, “Automotive, railway and avionics multicore systems”. [Online]. Available from: http://www.projekt-aramis.de/. Last access 2014.07.18.

[17] SPES_XT project, “Software platform embedded systems”. [Online]. Available from: http://spes2020.informatik.tu-muenchen.de/spes_xt-home.html. Last access 2014.07.18.

[18] A. Ploskonos and M. Uflacker, “A classification schema for process and method adaptation in software design projects”, Tenth International Design Conference (DESIGN 2008), May 2008, pp. 219-228.

[19] R. Firth, W. G. Wood, R. D. Pethia, L. Roberts and V. Mosley., "A classification scheme for software development methods", Technical Report CMU/SEI-87-TR-041, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1987.

[20] P. Reinkemeier, H. Hille, and S. Henkler, “Towards creating flexible tool chains for the design and analysis of multi-core systems”, Vierter Workshop zur Zukunft der Entwicklung softwareintensiver, eingebetteter Systeme (ENVISION 2020), colocated with Software Engineering 2014 conference, Feb. 2014. [Online]. Available from: http://ceur-ws.org/Vol-1129/paper37.pdf. Last access: 2014.07.21.

[21] RTCA DO-254, “Design Assurance Guidance for Airbone Electronic Hardware”, Apr. 2000.

[22] K. Forsberg and H. Mooz, “The Relationship of System Engineering to the Project Cycle”, First Annual Symposium of National Council on System Engineering, Oct. 1991, pp. 57-65.

[23] K. Schwaber and M. Beedle, “Agile software development with Scrum”, Prentice Hall, 2002, ISBN: 0-13-067634-9.

 ISBN: 978-1-61208-367-4

Working With Reverse Engineering Output for Benchmarking and Further Use

David Cutting and Joost Noppen

School of Computing Science
University of East Anglia
Norwich, Norfolk, UK
Email: {david.cutting,j.noppen}@uea.ac.uk

***Abstract*—Various tools exist to reverse engineer software source code and generate design information, such as UML projections. Each has specific strengths and weaknesses, however no standardised benchmark exists that can be used to evaluate and compare their performance and effectiveness in a systematic manner. To facilitate such comparison we introduce the Reverse Engineering to Design Benchmark (RED-BM), which consists of a comprehensive set of Java-based targets for reverse engineering and a formal set of performance measures with which tools and approaches can be analysed and ranked. When used to evaluate 12 industry standard tools performance figures range from 8.82% to 100% demonstrating the ability of the benchmark to differentiate between tools. Most reverse engineering tools can provide their output in the Extensible Metadata Information (XMI) format. Theoretically this should ensure tool interoperability but in practice the implementation of the XMI standard varies widely to the point where outputs cannot be exchanged between tools. In addition, this severely hinders the systematic usage of reverse engineering tool output, for example in a benchmark or for use in other analysis. To aid the comparison, analysis and further use of reverse engineering XMI output we have developed a parser which can interpret the XMI output format of the most commonly used reverse engineering applications, and is used in a number of tools. These tools offer the facility for standalone examination of one or more XMI files, comparison between outputs for benchmarking or measurement, the use of XMI within Eclipse to generate UML projections in UMLet, and use of reverse engineering output in combination with other sources of relationship information. Given the imperfect performance of the majority of the reverse engineering tools tested by the benchmark a future direction of research is the combination of different sources of information, multiple tool output or other data, to build a more complete and accurate picture of structural relationships within source code.**

Keywords–Reverse Engineering; Benchmarking; Tool Comparison; XMI; Software Comprehension; UML; UML Reconstruction.

I. INTRODUCTION

Reverse engineering is concerned with aiding the comprehensibility and understanding of existing software systems. With ever growing numbers of valuable but poorly documented legacy codebases within organisations reverse engineering has become increasingly important. In response, there are a wide number of reverse engineering techniques, which offer a variety in their focus from Unified Modelling Language (UML) projection to specific pattern recognition [1][2][3]. However, it is difficult to compare their effectiveness against each other, as no standard set of targets exist to support this goal over multiple approaches, a problem also found in the verification and validation of new tools and techniques [4]. Any performance evaluations which do exist are specific to an approach or technique. It is impossible, therefore to gain a comparative understanding of performance for a range tasks, or to validate new techniques or approaches. To address this gap, a benchmark of such targets, the Reverse Engineering to Design Benchmark (RED-BM) was created that can be used to compare and validate existing and new tools for reverse engineering.

The use of benchmarks as a means to provide a standardised base for empirical comparison is not new and the technique is used widely in general science and in computer science specifically. Recent examples where benchmarks have been successfully used to provide meaningful and repeatable standards include comparison of function call overheads between programming languages [5], mathematical 3D performance between Java and C++ [6], and embedded file systems [7]. Our benchmark provides the ability for such meaningful and repeatable standard comparisons in the area of reverse engineering.

Previous work reviewing reverse engineering tools has primarily focused on research tools many with the specific goal of identification of design patterns [2][3][8][9][10], clone detection [11] or a particular scientific aspect of reverse engineering, such as pattern-based recognition of software constructs [12]. A previous benchmarking approach for software reverse engineering focused on pattern detection with arbitary subjective judgements of performance provided by users [13]. The need for benchmarks within the domain of reverse engineering to help mature the discipline is also accepted [4].

To make further use of reverse engineering output, for example, between tools or for re-projection of UML, an Object Management Group (OMG) standard, the XML Metadata Interchange (XMI) format [14], is provided. XMI is a highly customisable and extensible format with many different interpretations. In practice tools therefore have a wide variation in their XMI output and exchange between reverse engineering tools, useful for interactive projection between tools without repetition of the reverse engineering process, is usually impossible. This variance in XMI format also hinders use of XMI data for further analysis outside of a reverse engineering tool, as individual tools are required for each XMI variation.

During the creation of the reverse engineering benchmark, two tools were developed which could analyse Java source code identifying contained classes, and then, check for the

 ISBN: 978-1-61208-367-4

presence of these classes within XMI output. Further work based upon the identification and analysis of variances within different reverse engineering tools' output, along with a desire to be able to integrate such output within more detailed analysis, led to the creation of a generic XMI parser (Section III). The parser solves the problem of XMI accessibility through generic use and abstract representation of structural data contained in XMI files of multiple formats. This parser is used by further tools for structural analysis or comparison as well as automated UML re-projection within Eclipse.

The remainder of this paper in organised as follows: in Section II, we introduce our benchmark, and show it's application to industry tools (Section II-E). Section III concerns our work to make further use of reverse engineering output through the development of a generic XMI Parser. Finally, Section IV summarises work to date and details our current and future direction of research.

II. The Reverse Engineering to Design Benchmark (RED-BM)

RED-BM facilitates the analysis of reverse engineering approaches based on their ability to reconstruct class diagrams of legacy software systems. This is accomplished by offering the source code of projects of differing size and complexity as well as a number of reference UML models. The benchmark provides a set of measures that facilitate the comparison of reverse engineering results, for example class detection, to reference models including a "gold standard" and a number of meta-tools to aid in the analysis of tool outputs.

The benchmark allows ranking of reverse engineering approaches by means of an overall performance measure that combines the performance of an approach with respect to a number of criteria, such as successful class or relationship detection. This overall measure is designed to be extensible through the addition of further individual measures to facilitate specific domains and problems. In addition the benchmark provides analysis results and a ranking for a set of popular reverse engineering tools which can be used as a yardstick for new approaches. Full details, models, targets, results as well as a full description of the measurement processes used can be found at [15]. Although based on Java source code, the concepts and measurements are applicable to any object-oriented language and the benchmark could be extended to include other languages.

A. Target Artefacts

Our benchmark consists of a number of target software artefacts that originate from software packages of varying size and complexity. The range of artefacts is shown in Table I where large projects are broken down into constituent components. In addition the Table contains statistics on the number of classes, sub-classes, interfaces and lines of code for each of the artefacts.

The benchmark artefact targets represent a range of complexity and architectural styles from standard Java source with simple through to high complexity targets using different paradigms, such as design patterns and presentation techniques. This enables a graduated validation of tools, as well as a progressive complexity for any new tools to test and assess their capabilities. Also, included within RED-BM are a set of *gold standards* for class and relationship detection against which tool output is measured. These standards were created by manual analysis supported by tools, as described in Section II-D.

TABLE I. Software Artefact Targets of the RED-BM

Software Target Artefact	**Main Classes**	**Sub Classes**	**Inter-faces**	**Lines of Code**
ASCII Art Example A				
Example A	7	0	0	119
ASCII Art Example B				
Example B	10	0	0	124
Eclipse				
org.eclipse.core. commands	48	1	29	3403
org.eclipse.ui.ide	33	2	6	3949
Jakarta Cactus				
org.apache.cactus	85	6	18	4563
JHotDraw				
org.jhotdraw.app	60	6	6	5119
org.jhotdraw.color	30	7	4	3267
org.jhotdraw.draw	174	51	27	19830
org.jhotdraw.geom	12	8	0	2802
org.jhotdraw.gui	81	29	8	8758
org.jhotdraw.io	3	2	0	1250
org.jhotdraw.xml	10	0	4	1155
Libre Office				
complex.writer	11	33	0	4251
org.openoffice.java. accessibility.logging	3	0	0	287
org.openoffice.java. accessibility	44	63	1	5749
All bundled code (sw + accessibility)	241	173	33	39896

Artefacts were chosen for inclusion on the basis that they provided a range of complexity in terms of lines of code and class counts, used a number of different frameworks, offered some pre-existing design information and were freely available for distribution (under an open-source licence). Two artefacts (ASCII Art Examples A and B) were created specifically for inclusion as a baseline offering a very simple starting point with full UML design and use of design patterns.

Cactus, although depreciated by the Apache Foundation, has a number of existing UML diagrams and makes use of a wide number of Java frameworks. Eclipse was included primarily owing to a very large codebase which contains a varied use of techniques. The large codebase of Eclipse also provides for the creation of additional targets without incorporating new projects. JHotDraw has good UML documentation available both from the project itself and some third-party academic projects which sought to deconstruct it manually to UML. As with Eclipse, Libre Office provides a large set of code covering different frameworks and providing for more targets if required.

B. Measuring Performance

RED-BM enables the systematic comparison and ranking of reverse engineering approaches by defining a set of *performance measures*. These measures differentiate the performance of reverse engineering approaches and are based on accepted quality measures, such as successful detection of classes and packages [16][17]. Although seemingly both trivial and essential within a reverse engineering tool, these measures provide a basic foundation for measurement to be built on, and represent the most common requirement in reverse engineering

 ISBN: 978-1-61208-367-4

for detection of structural elements. Further, as seen in Section II-E, these measures are alone capable of differentiating wide ranges of tool performance. The performance of tools with respect to a particular measure is expressed as the fraction of data that has been successfully captured. Individual measures are then used in conjunction to form a weighted compound measure of overall performance. In our benchmark we define three base measures to assess the performance of reverse engineering tools and approaches:

- **Cl:** The fraction of classes successfully detected
- **Sub:** The fraction of sub-packages successfully detected
- **Rel:** The fraction of relationships successfully detected

Each of these measures are functions that take a system to be reverse engineered *s* and a result *r* that is produced by a reverse engineering approach when applied to *s*. The formal definition of our three base measures are as follows:

$$Cl(s,r) = \frac{C(r)}{C(s)} \quad , \quad Sub(s,r) = \frac{S(r)}{S(s)} \quad , \quad Rel(s,r) = \frac{R(r)}{R(s)} \quad (1)$$

where

$C(x)$ is the number of correct classes in *x*

$S(x)$ is the number of correct (sub-)packages in *x*

$R(x)$ is the number of correct relations in *x*

The overall performance *P* of a reverse engineering approach for the benchmark is a combination of these performance measures. The results of the measures are combined by means of a weighted sum which allows users of the benchmark to adjust the relative importance of, e.g., class or relation identification. We define the overall performance of a reverse engineering approach that produces a reverse engineering result *r* for a system *s* as follows:

$$P(s,r) = \frac{w_{CL}CL(s,r) + w_{Sub}Sub(s,r) + w_{Rel}Rel}{w_{CL} + w_{Sub} + w_{Rel}} \quad (2)$$

In this function, w_{CL}, w_{Sub} and w_{Rel} are weightings that can be used to express the importance of the performance in detecting classes, (sub-)packages and relations respectively. The benchmark results presented in this article all assume that these are of equal importance: $w_{CL} = w_{Sub} = w_{Rel} = 1$.

C. Application of the Benchmark

To analyse the effectiveness of our benchmark, we apply a range of commercial and open source reverse engineering tools (shown in Table II) to each target artefact. Each of the tools is used to analyse target source code, generate UML class diagram projections (if the tool supports such projections) and export standardised XMI data files. Although the source code target artefacts used for testing are broken down into the package level for analysis, the reverse engineering process is run on the full project source code to facilitate package identification. The output produced by each of the tools is subsequently analysed and compared to the reference UML documentation using a benchmark toolchain we specifically created for comparison of class detection rates (see Section II-D). Finally, we perform a manual consistency between the standard tool output and XMI produced to identify and correct any inconsistencies where a tool had detected an element but not represented it within the generated XMI.

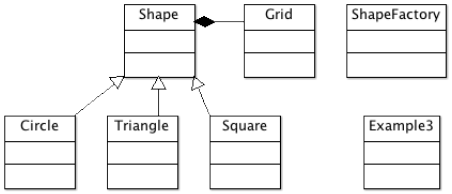


Figure 1. Reference Class Diagram Design for ASCII Art Example A

When analysing the results a wide range of variety can be observed even for simple targets such as Example A, one of the simplest targets with just 7 classes, as depicted in Figure 1. Please note that although Example A only contains generalisation and composition relationships other target artefacts contained associations, and these were included in the measurement. It can be seen in Figure 2 that Software Ideas Modeller failed to identify and display any relationship between classes. Other tools such as ArgoUML [18] (Figure 3) were very successful in reconstructing an accurate class diagram when compared to the original reference documentation.

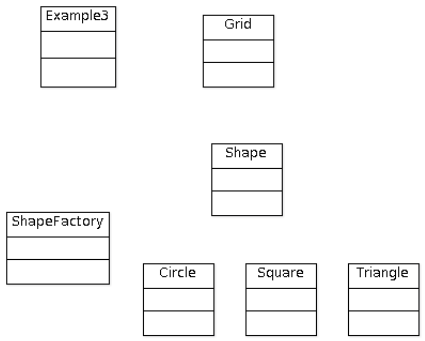


Figure 2. ASCII Art Example A Output for Software Ideas Modeller

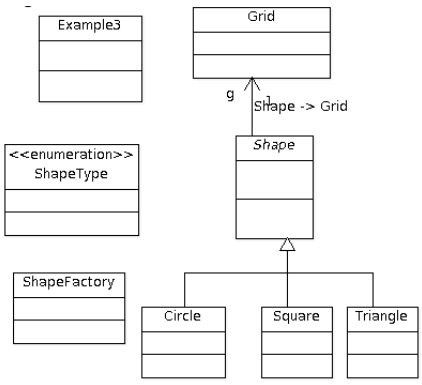


Figure 3. ASCII Art Example A Output for ArgoUML

In stark contrast to tools which performed well (e.g., Rational Rhapsody and ArgoUML) a number of tools failed

to complete reverse engineering runs of benchmark artefacts and even crashed repeatedly during this procedure. The result of which is that they are classified as detecting 0 classes for those target artefacts. While some tools failed to output valid or complete XMI data, a hindrance to their usability and ease of analysis, this has not affected their performance evaluation as their performance could be based on our manual analysis of their UML projection.

TABLE II. LIST OF TOOLS AND VERSIONS FOR USE IN EVALUATION

Tool Name (Name Used)	**Version Used (OS) Licence**
ArgoUML	0.34 (Linux) Freeware
Change Vision Astah Professional (Astah Professional)	6.6.4 (Linux) Commercial
BOUML	6.3 (Linux) Commercial
Sparx Systems Enterprise Architect (Enterprise Architect)	10.0 (Windows) Commercial
IBM Rational Rhapsody Developer for Java (Rational Rhapsody)	8.0 (Windows) Commercial
NoMagic Magicdraw UML (MagicDraw UML)	14.0.4 Beta (Windows) Commercial
Modeliosoft Modelio (Modelio)	2.2.1 (Windows) Commercial
Software Ideas Modeller	6.01.4845.43166 (Windows) Commercial
StarUML	5.0.2.1570 (Windows) Freeware
Umbrello UML Modeller (Umbrello)	2.3.4 (Linux) Freeware
Visual Paradigm for UML Professional (Visual Paradigm)	10.1 (Windows) Commercial
IBM Rational Rose Professional J Edition (Rational Rose)	7.0.0.0 (Windows) Commercial

D. Benchmark Toolchain

To facilitate effective analysis and ease reproduction or repetition of the results a toolchain was developed for use within RED-BM, consisting of two main components (*jcAnalysis* and *xmiClassFinder*), combined to measure the rate of class detection. The steps followed in the application of the benchmark are shown in Figure 4 with the developed tools highlighted.

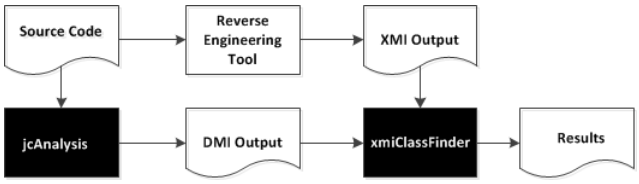


Figure 4. RED-BM Process with Toolchain Elements Highlighted

1) jcAnalysis: This tool recurses through a Java source tree analysing each file in turn to identify the package along with contained classes (primary and sub-classes). The list of classes is then output in an intermediate XML format (DMI). For every target artefact, jcAnalysis' output was compared against a number of other source code analysis utilities, including within Eclipse, to verify the class counts. A manual analysis was also performed on sections of source code to verify naming.

2) xmiClassFinder: This tool analyses an XMI file from a reverse engineering tool and attempts to simply identify all the classes contained within the XMI output (the classes detected by the reverse engineering tool in question). The classes contained within the XMI can be automatically compared to input from jcAnalysis (in DMI format) for performance (classes correctly detected) to be measured.

Once an analysis had been completed, a manual search was then performed on the source code, in XMI output, and within the reverse engineering tool itself, to try and locate classes determined as "missing" by the toolchain. This step also served to validate the toolchain, in that classes identified as "missing" were not then found to be actually present in the reverse engineering output.

E. Evaluation of Analysis Results

For the analysis of the results produced by the reverse engineering tools, we use a standard class detection performance measure for all targets (*CD*, formula 2).

To further refine the evaluation of the reverse engineering capabilities of approaches, we divide the artefacts of the benchmark into three categories of increasing complexity; C1, C2 and C3. These categories allow for a more granular analysis of tool performance at different levels of complexity. For example, a tool can be initially validated against the lowest complexity in an efficient manner only being validated against higher complexity artefacts at a later stage. Our complexity classes have the following boundaries:

- **C1:** $0 \leq$ number of classes ≤ 25
- **C2:** $26 \leq$ number of classes ≤ 200
- **C3:** $201 \leq$ number of classes

The complexity categories are based on the number of classes contained in the target artefact. As source code grows in size both in the lines of code and the number of classes it becomes inherently more complex, and so, more difficult to analyse [19][20]. While a higher number of classes does not necessarily equate to a system that is harder to reverse engineer, we have chosen this metric as it provides a quantitative measure without subjective judgement.

The bounds chosen for these categories demonstrated a noticeable drop-off in detection rates observed in many of the tools (Table III). However, any user of the benchmark can introduce additional categories and relate additional performance measures to these categories to accommodate for large scale industrial software or more specific attributes, such as design patterns.

Finally, we use the compound measure *CM*, which contains the three complexity measures with weighting as follows: $w_{C1} = 1, w_{C2} = 1.5, w_{C3} = 2$; giving a higher weighting to target artefacts that contain more lines of code.

Using these performance measures a wide range of results between the tools used for analysis can be seen. Some tools offer extremely poor performance, such as Rational Rose and Umbrello, as they crashed or reported errors during reverse engineering or UML projection, failing to detect or display classes and relationships entirely for some targets. As a general trend, the percentage of classes detected on average declined as the size of the project source code increased. As the number of classes detected varied significantly in different tools (Figure

 ISBN: 978-1-61208-367-4

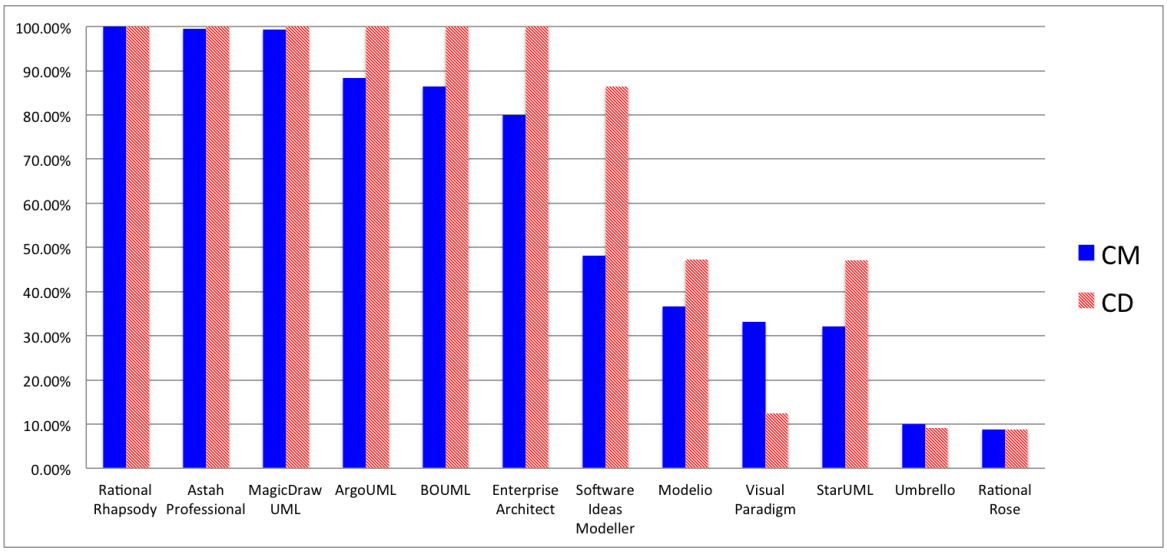


Figure 5. Overall Class Detection (CD) and Compound Measure (CM) Performance by Tool

TABLE III. CRITERIA RESULTS BY TOOL

Criterion > ∨ **Tool**	**CD** %	**C1** %	**C2** %	**C3** %	**CM** %
ArgoUML	100	98.15	75	100	88.27
Astah Professional	100	97.62	100	100	99.47
BOUML	100	92.59	75	100	86.42
Enterprise Architect	100	66.67	62.22	100	80.00
Rational Rhapsody	100	100	100	100	100.00
MagicDraw UML	100	98.15	100	100	99.38
Modelio	47.33	95.92	29.66	12.02	36.54
Software Ideas Modeller	86.41	62.15	41.48	46.04	48.10
StarUML	47.11	47.22	23.47	31.16	32.17
Umbrello	9.2	35.79	5.95	0	9.94
Visual Paradigm	12.42	38.18	51.68	16.67	33.12
Rational Rose	8.69	38.05	1.09	0	8.82

5) so did the amount of detected relationships, to a degree this can be expected as if a tool fails to find classes it would also fail to find relationships between these missing classes. In this figure the difference between the standard class detection measure CD and the compound measure CM becomes clear as, for example, ArgoUML was very strong in class detection but performed at a slightly lower level on relation detection, which is explicitly considered in the compound measure. It is also interesting to note that Visual Paradigm offered better performance for the compound measure as opposed to class detection highlighting its superior ability to deal with relations and packages as compared to class detection.

Overall our benchmark identified IBM Rational Rhapsody as the best performer as it achieved the maximum score for our compound measure (100%) with two other tools, Astah Professional and MagicDraw UML coming in a close second scoring in excess of 99%. As the poorest performers our work highlighted Umbrello, Visual Paradigm and notably IBM Rational Rose which scored the lowest with a compound measure of just 8.82% having only detected 8.69% of classes. A detailed breakdown of the performance of the tools for individual targets is provided with the benchmark [15].

III. XMI PARSER

As previously mentioned the XMI standard is highly fragmented and cannot be used as designed to interchange information between tools. It is also desirable to be able to make use of reverse engineering output for further use or analysis (for example, within a benchmark). Therefore, building from the knowledge gained in creating the toolchain for the benchmark, the simple *xmiClassFinder* tool, a XMI Parser was created.

This is a generic component designed for integration within other projects consisting of a Java package. The parser is capable of reading an XMI file, of most common output formats, recovering class and relationship information in a structured form. Data access classes are provided, which contain the loaded structural information, and can be accessed directly or recursively by third-party tools. As a self-contained utility package, the XMI Parser can be developed in isolation to tools making use of it and be incorporated into tools when required. A number of tools have been and continue to be developed within UEA to make use of reverse engineering information through implementation of the XMI Parser.

A. XMI Analyser

XMI Analyser uses the generic XMI Parser to load one or more XMI files which can then be analysed. Features include a GUI-based explorer showing the structure of the software and items linked through relationships. A batch mode can be used from the command line for automated loading of XMI files and analysis. XMI Analyser is primarily used for testing revisions to the XMI Parser, as an example application and also for the easy viewing of structural information contained within XMI, as shown in Figure 6.

XMI Analyser is also capable of comparison between multiple XMI files generating a report highlighting any differences found. This analysis can inform decisions as to the accuracy of the reverse engineering data represented in reverse engineering output.

 ISBN: 978-1-61208-367-4

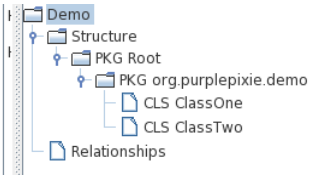


Figure 6. XMI Analyser Structure Display

B. Eclipse UMLet Integration

One of our desired outcomes was the ability to re-project UML outside of a specific reverse engineering tool. Such a capability would not only allow for detailed UML projections without access to the reverse engineering tool, but also programatic projection, for example in an interactive form. The Eclipse UMLet Integration, the interface of which is shown in Figure 7, is in the form of a plugin for the Eclipse Framework. The XMI Parser and supporting interfaces are included along with a graphical window-based interface and a visualisation component. This tool can load one or more XMI files and associate them with open or new UMLet documents. These documents can then be used to automatically generate a UML class diagram projection containing the structural elements contained within the XMI. An example of a re-projection within UMLet can be seen in Figure 8; please note, however, owing to a limitation in our UMLet API relationships are recovered but not shown.

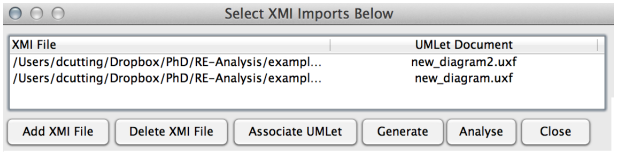


Figure 7. Eclipse Visualisation Interface

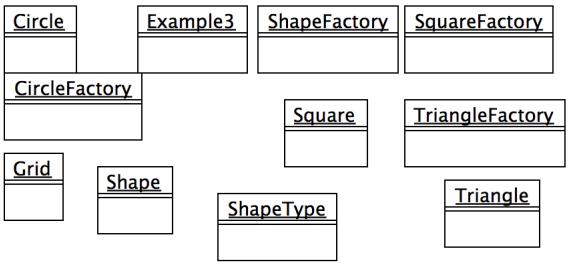


Figure 8. Eclipse UMLet Re-Projection of UML

C. Java Code Relation Analysis (jcRelationAnalysis)

The *jcRelationAnalysis* tool is a generic utility designed to analyse and comprehend the relationship between elements (classes) in Java source code. This is accomplished by first building a structural picture of the inter-relationships between elements, such as classes, contained within a source code corpus, initially from reverse engineering output, for which the XMI Parser is used. The ultimate intention of the tool is to work with combinational data from a number of different sources to compare or augment relationship information. This tool is now being used and further developed within our current and future research (Section IV).

IV. Conclusion and Future Direction

To analyse the effectiveness of RED-BM we applied it to a range of reverse engineering tools, ranging from open source to comprehensive industrial tool suites. We demonstrated that RED-BM offers complexity and depth as it identified clear differences between tool performance. In particular, using the compound measure (CM) RED-BM was capable of distinguishing and ranking tools from very low (8.82%) to perfect (100%) performance.

The XMI Parser allows tools to make direct use of reverse engineering output overcoming the fragmentation issues. The capability of direct use of reverse engineering output is clearly demonstrated through the ability for UML to be re-projected within UMLet, and also used in other tools for further analysis.

The future direction of our work will be to combine reverse engineering output with other sources of information about a source corpus, for example mining repository metadata or requirement documentation. The *jcRelationAnalysis* tool is being used as a programmable basis for integration of different sources of information into a common format of relationships between source code elements. These relationships, be they direct and found through reverse engineering, such as generalisations, or semantic in nature and found through other means, will be used in combination to form a more complete understanding of a software project.

Such analysis will aid both general comprehension of software and also change impact analysis by identifying relationships between elements not immediately obvious at the code or UML level.

References

[1] G. Rasool and D. Streitfdert, “A survey on design pattern recovery techniques,” International Journal of Computing Science Issues, vol. 8, 2011, pp. 251–260.

[2] J. Roscoe, “Looking forwards to going backwards: An assessment of current reverse engineering,” Current Issues in Software Engineering, 2011, pp. 1–13.

[3] F. Arcelli, S. Masiero, C. Raibulet, and F. Tisato, “A comparison of reverse engineering tools based on design pattern decomposition,” in Software Engineering Conference, 2005. Proceedings. 2005 Australian. IEEE, 2005, pp. 262–269.

[4] S. E. Sim, S. Easterbrook, and R. C. Holt, “Using benchmarking to advance research: A challenge to software engineering,” in Proceedings of the 25th International Conference on Software Engineering. IEEE Computer Society, 2003, pp. 74–83.

[5] A. Gaul, “Function call overhead benchmarks with matlab, octave, python, cython and c,” arXiv preprint arXiv:1202.2736, 2012.

[6] L. Gherardi, D. Brugali, and D. Comotti, “A java vs. c++ performance evaluation: a 3d modeling benchmark,” Simulation, Modeling, and Programming for Autonomous Robots, 2012, pp. 161–172.

[7] P. Olivier, J. Boukhobza, and E. Senn, “On benchmarking embedded linux flash file systems,” arXiv preprint arXiv:1208.6391, 2012.

[8] S. Uchiyama, H. Washizaki, Y. Fukazawa, and A. Kubo, “Design pattern detection using software metrics and machine learning,” in First International Workshop on Model-Driven Software Migration (MDSM 2011), 2011, p. 38.

[9] I. Philippow, D. Streitferdt, M. Riebisch, and S. Naumann, “An approach for reverse engineering of design patterns,” Software and Systems Modeling, vol. 4, no. 1, 2005, pp. 55–70.

[10] N. Pettersson, W. Lowe, and J. Nivre, “Evaluation of accuracy in design pattern occurrence detection,” Software Engineering, IEEE Transactions on, vol. 36, no. 4, 2010, pp. 575–590.

 ISBN: 978-1-61208-367-4

[11] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, "Comparison and evaluation of clone detection tools," Software Engineering, IEEE Transactions on, vol. 33, no. 9, 2007, pp. 577–591.

[12] M. Meyer, "Pattern-based reengineering of software systems," in Reverse Engineering, 2006. WCRE'06. 13th Working Conference on. IEEE, 2006, pp. 305–306.

[13] L. Fulop, P. Hegedus, R. Ferenc, and T. Gyimóthy, "Towards a benchmark for evaluating reverse engineering tools," in Reverse Engineering, 2008. WCRE'08. 15th Working Conference on. IEEE, 2008, pp. 335–336.

[14] OMG et al., "Omg mof 2 xmi mapping specification," http://www.omg.org/spec/XMI/2.4.1, 2011, [Online; accessed December 2012].

[15] UEA, "Reverse engineering to design benchmark," http://www.uea.ac.uk/computing/machine-learning/traceability-forensics/reverse-engineering, 2013, [Online; accessed May 2013].

[16] R. Koschke, "Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey," Journal of Software Maintenance and Evolution: Research and Practice, vol. 15, no. 2, 2003, pp. 87–109.

[17] G.-C. Roman and K. C. Cox, "A taxonomy of program visualization systems," Computer, vol. 26, no. 12, 1993, pp. 11–24.

[18] ArgoUML, "Argouml," http://argouml.tigris.org/, 2012, [Online; accessed December 2012].

[19] N. E. Fenton and S. L. Pfleeger, Software metrics: a rigorous and practical approach. PWS Publishing Co., 1998.

[20] B. Bellay and H. Gall, "A comparison of four reverse engineering tools," in Reverse Engineering, 1997. Proceedings of the Fourth Working Conference on. IEEE, 1997, pp. 2–11.

 ISBN: 978-1-61208-367-4

Software Relialibility Markovian Model Based on Phase-Type Distribution

Mindaugas Brazenas, Eimutis Valakevicius
Department of Mathematical Modeling
Kaunas University of Technology
Kaunas, Lithuania
e-mail: mindaugas.brazenas@yahoo.com; eimval@ktu.lt

Abstract— **The paper focuses on creating of a software reliability model based on phase type distribution. Usually, the length of intervals between the moments of fault detection and correction have unknown distributions. In this paper, a new approach how to approximate any distribution of positive random variable by mixture and convolution of exponential phases, known as the general type of phase-type distribution, is proposed. The optimization algorithm of Local Unimodal Sampling (LUS) is applied to estimate parameters of phase-type distribution. After such procedure, the dynamics of a software reliability model can be described by a continuous time absorbing Markov chain. The probabilities of the resulting absorbing Markov chain are used to compute performance measures of the software reliability model.**

Keywords-software reliability model; phase–type distribution; absorbing Markov chain; performance measures.

I. INTRODUCTION

Software reliability is the failure probability of the software under investigation. The situation on creating software reliability models is clearly explained in the following citation from the Wikipedia: "Over 225 models have been developed since early 1970s, but how to quantify reliability still remains unsolved. There is no single model which can be used in every situation. There is no model which is either complete or fully developed" [1].

A software reliability model allows forecasting the software reliability at any moment of time. One of the important problems in creating models is an assumption about distribution of the length of intervals between the moments of fault detection. Some of authors assumes that the length of intervals is distributed according to the exponential law [2][3]. For example, the model developed by Moranda and Jelinski [4] assumes an exponential time between failures having parameter that time intervals of detection software faults follow exponential law with the parameter proportional to the number of faults remaining in the system. The similar assumptions are used in [5][6]. Recently, non-homogenous Poisson processes became popular for describing stochastic behavior of the number of detected faults, because of their simplicity [7][8][9]. Beside the mentioned distributions, other models that are based on Weibull [10], hyper geometric [11], Pareto [12] and other distributions [13] are investigated.

The use in the software reliability model of any non-exponential distributions is complicated from the computing point of view. Therefore, in this paper, a novel approach to apply a convolution and mixture of exponential distributions, called the Phase-Type (PH) distribution, to approximate time distributions of fault detection and fixing is suggested. It is known that the PH distribution can approximate an arbitrary probability distribution of a positive random variable with an arbitrary accuracy by adjusting the phase structure [14]. Some authors use concrete structure of PH distributions, such as Erlang and hyper exponential [15], Cox [16], or others. The concrete structure of the PH distribution may not approximate the desired distribution with the required accuracy. Many models have been utilised for evaluating the quality of a software using reliability but very little focus on general type of three phase distribution. Hence, this paper mainly focuses on this direction. Using such distribution, the performance of the model can be described by an absorbing Markov chain [14].

The paper is organized as follows. Section II gives description of software reliability model under consideration. Section III describes the algorithm for finding the structure and parameters of approximating PH distribution. The algorithm for constructing the set of all possible states of the system and transition matrix between states is given in Section IV. The modelling results are presented in Section V. The paper is concluded in Section VI.

II. DESCRIPTION OF THE MODEL

Let us describe the conceptual model of a software reliability model. Say, that the software contains a fixed number of faults Fc (fault count). Assume that the fault detection time follows some distribution $F^{(a)}(x)$ and fixing time of detected faults obeys another distribution law $F^{(b)}(x)$. The modelling process can be represented as the queuing system (see Figure 1).

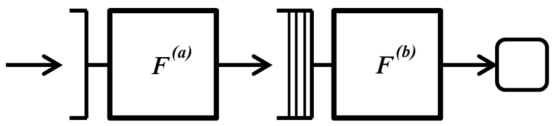


Figure 1. The process of identifying and fixing faults in software.

It is proposed to approximate any general distribution of a positive random variable by the general phase-type distribution (GPH). The phase-type (PH) distribution is defined as the absorbing time distribution of Continuous-Time Markov Chain (CTMC). The detected fault enters the

 ISBN: 978-1-61208-367-4

queue if the previously detected one is not fixed yet. The process of detecting and fixing faults ends when all the faults are identified and fixed. The developed software reliability model gives probabilistic measures of the process.

III. PARAMETER ESTIMATION OF THE PHASE-TYPE DISTRIBUTIONS

Parameter estimation of general phase-type distribution is one of the most challenging problems.

The precision of approximation of non-markovian model M by markovian one M^* depends on how well distributions $F^{(a)}$, $F^{(b)}$ are approximated by phase-type distributions $PH^{(a)}$, $PH^{(b)}$. There are several methods to search for optimal phase-type distribution parameters: moment matching method [17][18], expectation maximization method [19][20][21][22], and others. We will search for the optimal parameters by employing a vector optimization algorithm.

The phase-type distribution $PH(\boldsymbol{\pi}, T)$, which has three exponential phases (see Figure 2),

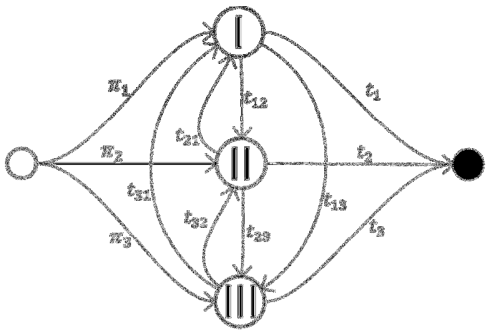


Figure 2. The general structure of PH distribution with three phases.

is determined by 12 variables :

$$\pi = [\pi_1, \pi_2, \pi_3], T = \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{bmatrix} \quad (1)$$

The coordinate $\pi_i, i = 1,2,3$ of the vector $\boldsymbol{\pi}$ (1) denotes the probability of process starting in ith phase. The intensity rates of transition from one phase to other are defined in matrix T. For example, the value t_{13} indicates the average transition number from the first phase to the third one per unit of time. The auxiliary vector $\boldsymbol{t} = [t_1, t_2, t_3] : \boldsymbol{t} = (I - T)\mathbf{1}$ denotes rates the absorbing state (black circle) from each phase is reached. For example, the value t_2 indicates the rate of transition from second phase to absorbing state. The transition rates satisfy the following equalities and unequalities

$$\begin{gathered} t_{11}, t_{22}, t_{33} < 0, \\ t_{12}, t_{13}, t_{21}, t_{23}, t_{31}, t_{32}, t_1, t_2, t_3 \geq 0, \\ t_{11} = -(t_{12} + t_{13} + t_1), \\ t_{22} = -(t_{21} + t_{23} + t_2), \\ t_{33} = -(t_{31} + t_{32} + t_3). \end{gathered} \quad (2)$$

The problem of finding optimal parameters of $PH(\boldsymbol{\pi}, T)$ is transformed to a problem of finding the vector $\boldsymbol{x}^* \in [\boldsymbol{l}, \boldsymbol{u}]$, such that $\forall \boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}] : g(\boldsymbol{x}^*) \leq g(\boldsymbol{x})$. Here $g(\cdot)$ is an objective function; $\boldsymbol{l}, \boldsymbol{u}$ – lower and upper bounds of vector $\boldsymbol{x}$. The mapping of vector $\boldsymbol{x}$ to the set of parameters of the phase-type distribution $PH(\boldsymbol{\pi}, T)$ is carried out in the following way

$$\begin{gathered} \boldsymbol{x} = (x_1, x_2, \dots, x_{12}) \to \boldsymbol{\pi} := \frac{[x_1, x_2, x_3]}{x_1 + x_2 + x_3}, \\ t_1 := x_4, t_{12} := x_5, t_{13} := x_6, \\ t_{21} := x_7, t_2 := x_8, t_{23} := x_9, \\ t_{31} := x_{10}, t_{32} := x_{11}, t_3 := x_{12}. \end{gathered} \quad (3)$$

The objective function, to be minimized, is defined as an area between the density functions $f(x; \boldsymbol{\theta})$ and $f_{PTD}(x; \boldsymbol{\pi}, T)$, as

$$S = \int_0^\infty |f_{PH}(x; \boldsymbol{\pi}, T) - f(x; \boldsymbol{\theta})| \, dx \quad (4)$$

The estimation of S (4) is obtained by the following expression

$$\hat{S}(x_{end}, \Delta x; \boldsymbol{\pi}, T) = \sum_{k=1}^{n} |f_{PH}(x_k; \boldsymbol{\pi}, T) - f(x_k; \boldsymbol{\theta})| \Delta x,$$

$$x_k = (k - 0.5)\Delta x, \; n = \left\lfloor \frac{x_{end}}{\Delta x} \right\rfloor \quad (5)$$

where: x_{end} denotes the end value of discretization and Δx – the step of discretization. After the discretization the objective function (4) has the form

$$g(\boldsymbol{x}) := \hat{S}(x_{end}, \Delta x; \boldsymbol{\pi}, T); \; \boldsymbol{x} \to \boldsymbol{\pi}, T. \quad (6)$$

The lower and upper bounds of $\boldsymbol{x}$ are defined as

$$\begin{gathered} \boldsymbol{l} = \left[\underbrace{0,0, \dots, 0}_{12}\right], \\ \boldsymbol{u} = \left[1,1,1, \underbrace{\lambda_{max}, \lambda_{max}, \dots, \lambda_{max}}_{9}\right], \end{gathered} \quad (7)$$

where λ_{max} – is the maximal transition rate. Using the mapping $\boldsymbol{x} \to \boldsymbol{\pi}, T$, the optimal parameters of the phase-type distribution are obtained from solution $\boldsymbol{x}^*$, which is given by a certain optimization algorithm.

 ISBN: 978-1-61208-367-4

IV. SYSTEM STATE GRAPH CONSTRUCTION ALGORITHM

The scheme of the process of detecting and fixing faults in software after approximating the arbitrary distributions by PH distributions is represented in Figure 3.

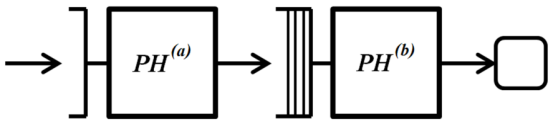


Figure 3.The process of detecting and fixing faults in software after distribution approximation.

The algorithm for constructing the set of states and the transition matrix for markovian model M^* **is** described in this section. The set of states of the system is defined by the vector $\boldsymbol{v} = (k_1, k_2, k_3, k_4)$, where $k_1 \in \{1,2,3\}$ denotes the index of an active phase of $PH^{(a)}$ and $k_1 = 0$ indicates that there is any active phase in $PH^{(a)}$; $k_2 \in \{0,1, \dots, Fc-1\}$ denotes a number of detected faults which are waiting in the queue; $k_3 \in \{1,2,3\}$ denotes the index of an active phase of $PH^{(b)}$, $k_3 = 0$ indicates that there is any active phase in $PH^{(b)}$;$k_4 \in \{0,1, \dots, Fc\}$ – denotes a number of fixed faults. The number of detected faults in the state $\boldsymbol{v}$ is defined according the formula

$$c(\boldsymbol{v}) = \mathbf{1}_{\{k_1>0\}} + k_2 + \mathbf{1}_{\{k_3>0\}} + k_4. \quad (8)$$

All the states are enumerated by the mapping

$$I_v = \omega(\boldsymbol{v}) \in \{0,1, \dots, r_{max}-1\}, \quad (9)$$

where r_{max} denotes the number of states: $r_{max} = 16Fc(Fc+1)$. The mapping $\omega(\boldsymbol{v})$ is defined as

$$\omega(\boldsymbol{v}) = (Fc+1)(4k_1Fc + 4k_2 + k_3) + k_4. \quad (10)$$

The inverse mapping $\omega^{-1}(I_v)$ is obtained by the formulas

$$\omega^{-1}(I_v) = \big(c, b\%Fc, a\%4, I_v\%(Fc+1)\big), \quad (11)$$

where: $a = \frac{I_v - I_v\%(Fc+1)}{Fc+1}$, $b = \frac{a-a\%4}{4}$, $c = \frac{b-b\%Fc}{Fc}$, and % is a reminder operator.

The vector $\boldsymbol{u} = \big(u_0, u_1, \dots, u_{r_{max}-1}\big)$ of boolean variables (false or true) are used to determine all the possible states of the system. The following sets of states are used: V^{init} contain**s** initial states of the system; V contain**s** all possible states of the system; V^{np} include**s** states which are going to be investigated in the next iteration and V^{tmp} is the temporary set of states obtained from the investigated states after one iteration. Each state contained in these sets is represented by its index I_v.

The transitions rates between all states are stored in the matrix $Q^+ = \{q_{ij}^+\};\ i,j = \overline{0, r_{max}-1}$.

The algorithm for generating the set of possible states of the system consists of the following steps:

1) *Mark all the states as not investigated:*

$$u_{I_v} := false\ (\textit{the state is not investigated yet}),$$
$$I_v = \overline{0, r_{max}-1}$$

2) *Calculate the initial probability vector :*

$$\pi_{I_v}^+ := 0, I_v = \overline{0, r_{max}-1}$$
$$\pi^+_{\omega((i,0,0,0))} := \pi_i^{(a)}, i = 1,2,3$$

3) *Determine the initial states of the system:*

$$i = 1,2,3: \pi_i^{(a)} > 0 \Rightarrow V^{init} := V^{init} \cup \omega\big((i,0,0,0)\big)$$

4) *Determine the initial values of the sets:*

$$V := V^{init}, V^{tmp} := V^{init}$$
$$\{q_{ij}^+\} := 0;\ i,j = \overline{0, r_{max}-1}$$

5) *Determine the states that have to be investigated:*

$$V^{np} := V^{tmp}$$

Clear the temporary set $V^{tmp} := \emptyset$

Let $\{V^{np}\}_p$ be an p-th element of the set V^{np}.

$$p := 1$$

6) *Find the coordinates of the state vector to be investigated:*

$$\boldsymbol{v} := \omega^{-1}\big(\{V^{np}\}_p\big),\ I_v := \omega(\boldsymbol{v})$$

7) *Let us denote the set of the state vectors*, that can be reached from the state $\boldsymbol{v}$, by $V^* := \emptyset$ and the set of transition rates, at which all states in set V^* are reached from the state $\boldsymbol{v}$, by $\lambda^* = \emptyset$.

8) *Find the elements of the sets* $V^* = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_k\}$ and $\lambda^* = \{\lambda_1, \lambda_2, \dots, \lambda_k\}$. The algorithm will be described later.

9) *Update the transition rates matrix:*

$$q^+_{I_v I_{v_i}} := \lambda_i;\ I_{v_i} = \omega(\boldsymbol{v}_i), i = \overline{1,k}$$

10) *Include not yet investigated states into the sets* V and V^{tmp} (see Figure 4)

```
for i from 1 to k
  I_{v_i} := ω(v_i)
```

```
    if u_{I_{v_i}} = false then
      V^{tmp} := V^{tmp} ∪ {I_{v_i}}, V := V ∪ {I_{v_i}}
    endif
  endfor
```

Figure 4. Pseudocode for including not investigated states into the sets V and V^{tmp}.

11) Mark the current state $\boldsymbol{v}$ as investigated:

$$u_{I_v} := true$$

12) If $p < card(V^{np})$ then $p := p + 1$ and go to step 6.

13) If $V^{tmp} \neq \emptyset$ go to step 5.

14) Create the final transition rates matrix $Q = \{q_{ij}\}$ and the initial probability vector $\boldsymbol{\pi}(0)$ from Q^+ and $\boldsymbol{\pi}^+(0)$ (see Figure 5).

```
for i from 0 to card(V) − 1
  I := ω({V}_i),
  π_i := π_I^+
  for j from 0 card(V) − 1
    J := ω({V}_j),
    q_ij := q_IJ^+
  endfor
endfor
```

Figure 5. Pseudocode for creating final transition rates matrix and initial probability vector.

15) The end of the algorithm.

The explanation of the step 8 in detail follows. Denote by the vector $\boldsymbol{v} = (k_1, k_2, k_3, k_4)$ the state of the system. There are four events that make the system to change the state: e_1 – the change of an active phase in $PH^{(a)}$, e_2 – the detection of fault, e_3 – the change of an active phase in $PH^{(b)}$ and e_4 – the correction of fault. The pseudocode needed to process these events is shown in the Figures 6, 7, 8, and 9.

```
for i from 1 to 3
  if k_1 > 0, t_{k_1 i}^{(a)} > 0 then
    V* := V* ∪ {(i, k_2, k_3, k_4)},
    λ* := λ* ∪ {t_{k_1 i}^{(a)}}
  endif
endfor
```

Figure 6. Pseudocode for processing the e_1 event.

```
if k_3 = 0 then
  if c(v) < Fc then
    for j from 1 to 3
      for s from 1 to 3
        if k_1 > 0, t_{k_1}^{(a)} > 0, π_j^{(b)} > 0, π_s^{(a)} > 0 then
          V* := V* ∪ {(s, 0, j, k_4)},
          λ* := λ* ∪ {t_{k_1}^{(a)} π_j^{(b)} π_s^{(a)}}
        endif
      endfor
    endfor
  else
    for j from 1 to 3
      if k_1 > 0, t_{k_1}^{(a)} > 0, π_j^{(b)} > 0 then
        V* := V* ∪ {(0, 0, j, k_4)},
        λ* := λ* ∪ {t_{k_1}^{(a)} π_j^{(b)}}
      endif
    endfor
  endif
elseif k_3 > 0 then
  if c(v) < Ec then
    for s from 1 to 3
      if k_1 > 0, t_{k_1}^{(a)} > 0, π_s^{(a)} > 0 then
        V* := V* ∪ {(s, k_2 + 1, k_3, k_4)},
        λ* := λ* ∪ {t_{k_1}^{(a)} π_s^{(a)}}
      endif
    endfor
  else
    if k_1 > 0, t_{k_1}^{(a)} > 0 then
      V* := V* ∪ {(0, k_2 + 1, k_3, k_4)},
      λ* := λ* ∪ {t_{k_1}^{(a)}}
    endif
  endif
endif
```

Figure 7. Pseudocode for processing the e_2 event.

```
for i from 1 to 3
  if k_3 > 0, t_{k_3 i}^{(b)} > 0 then
    V* := V* ∪ {(k_1, k_2, i, k_4)},
    λ* := λ* ∪ {t_{k_3 i}^{(b)}}
  endif
endfor
```

Figure 8. Pseudocode for processing the e_3 event.

```
if k_2 = 0 then
  if k_3 > 0, t_{k_3}^{(b)} > 0 then
    V* := V* ∪ {(k_1, 0, 0, k_4 + 1)},
    λ* := λ* ∪ {t_{k_3}^{(b)}}
  endif
else
  for s from 1 to 3
    if k_3 > 0, t_{k_3}^{(b)} > 0, π_s^{(b)} > 0 then
      V* := V* ∪ {(k_1, k_2 − 1, s, k_4 + 1)},
      λ* := λ* ∪ {t_{k_3}^{(b)} π_s^{(b)}}
    endif
  endfor
endif
```

Figure 9. Pseudocode for processing the e_4 event

V. MODELING RESULTS

Assume that a software program contains 10 faults (Fc=10) and suppose that the length of intervals between the moments of fault detection has the following Weibull density function

$$f^{(a)}(x) = \frac{4}{3}\left(\frac{4}{3}\right)^{-0.2} e^{-\frac{x}{0.6}},\ x \geq 0. \quad (12)$$

The distribution of the length of intervals between the moments of fixing faults has the following Weibull density function

$$f^{(b)}(x) = \frac{1.3}{1.5}\left(\frac{1.3}{1.5}\right)^{0.3} e^{-\frac{x}{1.3}},\ x \geq 0. \quad (13)$$

The discretization parameters **are** $x_{end} = 8$, $\Delta x = 0.0625$.

The distributions $f^{(a)}, f^{(b)}$ are approximated by the phase-type distributions $f_{PH}^{(a)}(x;\ \boldsymbol{\pi}^{(a)}, T^{(a)})$, $f_{PH}^{(b)}(x;\ \boldsymbol{\pi}^{(b)}, T^{(b)})$ with three phases. The following optimal parameters for $f_{PH^{(a)}}(x;\ \boldsymbol{\pi}^{(a)}, T^{(a)})$ and $f_{PH}^{(b)}(x;\ \boldsymbol{\pi}^{(b)}, T^{(b)})$ density functions are estimated using the optimization algorithm LUS [23] and parameter mapping given in Section III.

$$\hat{S}^{(a)} = 0.012693,\ \boldsymbol{\pi}^{(a)} \approx [3.456 \cdot 10^{-4}, 0.767, 0.233],$$

$$T^{(a)} \approx \begin{bmatrix} -1.516 & 0.002 & 1.509 \\ 0.584 & -2.348 & 0.005 \\ 1.782 & 7.819 & -15.604 \end{bmatrix},$$

$$\hat{S}^{(b)} = 0.010634,\ \boldsymbol{\pi}^{(b)} \approx [0.008, 1.397 \cdot 10^{-4}, 0.992],$$

$$T^{(b)} \approx \begin{bmatrix} -1.236 & 0 & 0 \\ 1.913 & -16.338 & 6.770 \\ 0.897 & 0.331 & -1.402 \end{bmatrix}$$

The comparision of means and standard deviations between original and approximated distributions are given in the Tables 1 and 2.

TABLE I. MEAN AND STANDARD DEVIATION OF ORIGINAL, DISCRETIZED AND APPOXIMATING DISTRIBUTIONS OF TIME FOR FAULT DETECTION

	Distribution $F^{(a)}$	**Discretized distribution $\hat{F}^{(a)}$(deviation,%)**	**Phase-type distribution $PH^{(a)}$(deviation,%)**
Mean	0.6798	0.6769 (0.43%)	0.6958 (2.35%)
Standard deviation	0.8569	0.8364 (2.39%)	0.8588 (0.22%)

TABLE II. MEAN AND STANDARD DEVIATION OF ORIGINAL, DISCRETIZED AND APPOXIMATING DISTRIBUTIONS OF TIME FOR FAULT CORRECTION

	Distribution $F^{(b)}$	**Discretized distribution $\hat{F}^{(b)}$(deviation,%)**	**Phase-type distribution $PH^{(b)}$(deviation,%)**
Mean	1.3854	1.3841 (0.09%)	1.4000 (1.05%)
Standard deviation	1.0747	1.0719 (0.26%)	1.1126 (3.53%)

The graphs of the original and approximated density functions are represented in Figures 10 and 11.

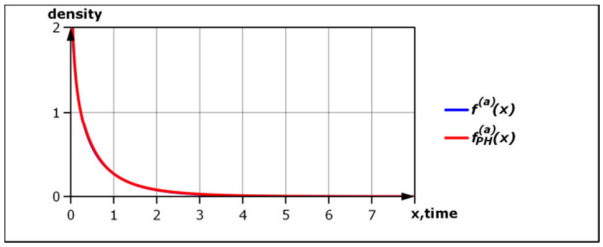


Figure 10. The density functions $f^{(a)}$ (blue) and $f_{PH}^{(a)}$ (red).

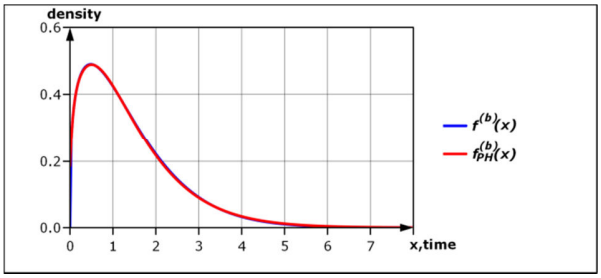


Figure 11. The density functions $f^{(b)}$ (blue) and $f_{PH}^{(b)}$ (red).

The markovian software reliability model has 466 states. The state probabilities are computed by the following formula

$$\boldsymbol{\pi}(t) = \boldsymbol{\pi}(0)e^{Qt}. \quad (14)$$

All possible states are grouped according the number of detected faults that are placed in the queue. The values $L_n(t)$, that there is a certain number n of faults waiting in the queue, are obtained by probability summation within each state group

$$L_n(t) = \sum_{i,k_2=n} \pi_i(t),$$
$$\omega^{-1}(\{V\}_i) = (k_1, k_2, k_3, k_4), n = \overline{0, Fc-1}, \quad (15)$$

where $\{V\}_i$ is the i^{th} element in the set V. The graph of the values $L_n(t)$ is shown in Figure 12.

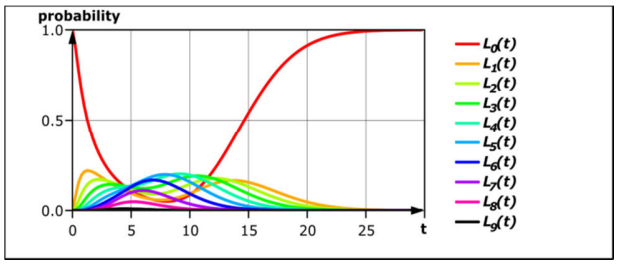


Figure 12. The probability functions (of time) of certain number of faults waiting in the queue.

Similarly, all possible states of the system are grouped according the number of fixed faults. The values $E_n(t)$, that there is a certain number n of fixed faults, are obtained by probability summation within each state group

$$E_n(t) = \sum_{i,k_4=n} \pi_i(t),$$
$$\omega^{-1}(\{V\}_i) = (k_1, k_2, k_3, k_4), n = \overline{0, Fc}, \quad (16)$$

where $\{V\}_i$ is the i^{th} element in the set V.
The graph of values $E_n(t)$ is shown in Figure 13.

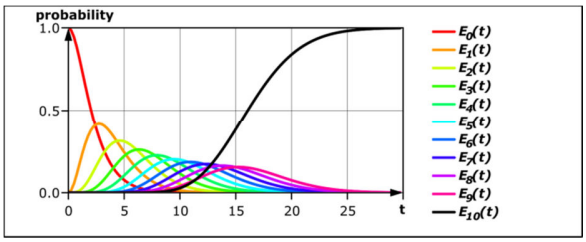


Figure 13. The probabilitiy functions (of time) of a certain number of fixed faults.

The density function of time of fixing all errors is shown in Figure 14.

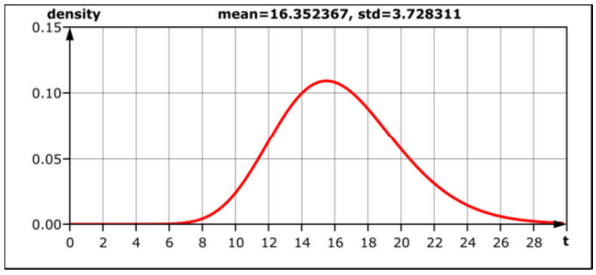


Figure 14. Density function of distribution of time necessary to fix all 10 faults.

The most probable that the time needed to fix all 10 faults is about 16 units of time.

VI. Conclusion and Future Works

In this article, a continuous time absorbing Markov chain model of software reliability was proposed. Non-markovian distributions of the length of intervals between the moments of fault detection and correction are approximated by the general phase-type distributions with three phases. The model generalizes other software reliability models in which various types of distributions are used. The probabilistic measures of detecting and fixing faults of created software are presented. The proposed model can be useful in estimating and monitoring software reliability, which is viewed as a measure of software quality. Therefore, it can be concluded that this model is more realistic then others for a detection of software faults.

In the future, the following modified software reliability model will be created and investigated. The detected fault must be fixed before searching for the next one with the assumption that the distributions can change depending on number of detected/fixed faults. Examples of the application how a model help to have better software will be added.

References

[1] List, "List of software reliability models", available: http://en.wikipedia.org/wiki/List_of_software_reliability_models [retrieved: July, 2014].

[2] B. Zachariah and R. N. Rattihalli, "Failure size proportional models and an analysis of failure detection abilities of software testing strategies," IEEE Transactions on Reliability, vol. 56, n. 2, 2007, pp. 246-253.

[3] Y. P. Wu, Q. P. Hu, M. Xie, and S. H. Ng, "Modeling and analysis of software fault detection and correction process by considering time dependency," IEEE Transactions on Reliability, vol. 56, n. 4, 2007, pp. 629-642.

[4] P. Moranda and Z. Jelinski, "Final report on software reliability study, MADC report number 63921," McDonnell Douglas Astronautics Company, 1972.

[5] J. Musa, A. Iannino, and K. Okumoto, "Software reliability measurement, prediction, application," McGraw Hill, NewYork, 1987.

[6] A. Goel and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance

measures," IEEE Transactions on Reliability, R–28 (3), 1979, pp. 206–211.

[7] A. L. Goel, "Software reliability models: assumptions, limitations and applicability," IEEE Trans. Software Eng., SE-11, 1985, pp. 1411–1423.

[8] S. S. Gokhale and K. S. Trivedi, "Log-logistic software reliability growth model," Proc. 3rd IEEE Int'l. High-Assurance Systems Eng., Symp IEEE CS Press, 1998, pp. 34–41.

[9] M. Ohba, "Inflection S-shaped software reliability growth model," Stochastic Models in Reliability Theory, (S. Osaki and Y. Hatoyama, eds.), Springer-Varlag, Berlin, Germany, 1984, pp. 144–165.

[10] S. Quadri and N. Ahmad, "Software reliability growth modeling with new modified Weibul testing-effort and optimal release policy," International Journal of Computer Applications, Vol. 6, No.2, 2010, pp. 1-10.

[11] Y. Tohma, R. Jacoby, Y. Murata, and M. Yamamoto, "Hypergeometric distribution model to estimate the number of residual software faults," Proceedings of the 13th Annual International Computer Software and Applications, Conference (COMPSAC'89), 1989, pp. 610-617.

[12] Y. Aamsidhar, Y. Srinivas, and A. Brahmini, "Software reliability growth model based on Pareto type III distribution," International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, Vol. 2, No.6, 2013**,** pp. 2694-2698.

[13] S. Inoue and S. Yamada, "Lognormal process software reliability modeling with testing-effort," Journal of Software Engineering and Application, Vol. 6, 2013**,** pp. 8-14.

[14] M. F. Neuts, "Matrix-Geometric Solutions in Stochastic Models: an Algorithmic Approach", Dover Publications Inc., 1981.

[15] H. Okamara and T. Dohi, "Building Phase-Type software reliability models', 17th International Symposium on Software Reliability Engineering (ISSRE'06), November 2007, pp. 289-298.

[16] V. Bubnov, A. Tyrva, and A. Khomonenko, "Model of reliability of the software with Coxian distribution of length of intervals between the moments of detection errors," IEEE 35th Annual Computer Software and Applications Conference Workshops, COMPSACW, 2011, pp. 310-314.

[17] A. Bobbio, A. Horváth, and M. Telek, "Matching three moments with minimal acyclic phase type distributions," Stochastic Models, 21, 2005, pp. 303-326.

[18] H. András and T. Miklós, "Matching More Than Three Moments with Acyclic Phase Type Distributions," Stochastic Models, 23, 2007, pp. 167-194.

[19] S. Asmussen, O. Nerman, and M. Olsson, "Fitting Phase-Type Distributions via the EM Algorithm," Scandinavian Journal of Statistics, Vol. 23, No. 4, December 1996, pp. 419-441.

[20] R. Sadre and B. R. Haverkort, "Fitting Heavy-Tailed HTTP Traces with the New Sratified EM-Algoritm," IT-NEWS 2008 – 4th International Telecommunication Networking Workshop on QoS Multiservice IP Networks, 2008, pp. 256-261.

[21] A. Risha, V. Diev, and E. Smirni, "An EM-based technique for approximating long-tailed data sets with PH distributions," Performance Evaluation, 55 (2), 2004, pp. 147–164.

[22] L. J. R. Esparza, "Maximum likelihood estimation of pahse-type distributions," Kongens Lyngby 2010, IMM-PHD-2010-245.

[23] E. Pedersen and A. J. Chipperfield, "Local Unimodal Sampling", Hvass Laboratories Technical Report no. HL0801, 2008.

 ISBN: 978-1-61208-367-4

Vergil: Guiding Developers Through Performance and Scalability Inferno

Christoph Heger*, Alexander Wert* and Roozbeh Farahbod†
*Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany
Email: {christoph.heger, alexander.wert}@kit.edu
†SAP AG, 76131 Karlsruhe, Germany
Email: roozbeh.farahbod@sap.com

***Abstract*—Software performance problems, such as high response times and low throughput, are visible to end users and can have a significant impact on the user experience. Solving performance problems is an error-prone and time-consuming task that is ideally done with the help of experienced performance experts. They often provide solutions in the form of work activities to developers such as to move functionality from one component to another in order to solve performance problems. Existing approaches are mostly model-based and mainly neglect the code base and measurement-based techniques. They can miss important details from the implementation, configuration and deployment environment of the application. In this paper, we propose a novel approach in the field of software performance engineering with the goal to solve recurring performance and scalability problems based on a systematic process and formalization of expert knowledge. Starting with a set of detected performance problems in the target system, our proposed approach supports developers by identifying, evaluating and ranking of solutions, and by providing a work plan sketching the implementation of the selected solution. In an example with a Java EE application, we show the solution of a software bottleneck through result caching.**

Keywords–Software Performance; Software Engineering; Software Measurement; Performance Evaluation.

I. INTRODUCTION

Over the last decade, software performance practitioners have been documenting recurring performance problems and their identified root causes and solutions as performance anti-patterns (for example in [1][2]). The documentations include general definitions and solutions to the performance problems but nevertheless, solving recurring performance problems is still a manual and time-consuming task that requires expertise in software performance engineering [3], rigorous performance evaluation techniques [4], and a deep understanding of the system under study. After the presence of a performance problem has been observed and the root cause (or root causes) has been identified with the help of a performance expert, the solution process often includes a comprehensive analysis of the solution space and usually consists of (1) identification of possible changes, (2) evaluation of the performance impact of each possible change on the particular system, (3) effort estimation for applying one or more changes, and (4) deciding what changes to be applied to the system.

Currently, to the best of our knowledge, there are no approaches that help developers with the implementation of a performance and scalability solution with providing an ordered set of work activities at the code level. Existing approaches for solving performance problems, for example [5][6][7][8], are model-based. A shortcoming of model-based approaches is that not all performance problems can be solved at the model level [7] when the implementation, measurement-based experiments and monitoring-driven testing techniques are neglected. Additionally, cost factors and constraints have to be taken into account when a variety of solution choices exists. Furthermore, decision support mechanisms have to be integrated in order to support developers in selecting the most appropriate solution [9]. Only [8] considers using the effort estimation of the designer for the necessary design model changes in selecting a solution among alternatives. Jing Xu also uses the determined changes to suggest what should be changed in an abstract way, but not how to do it concretely [8]. Nevertheless, the existing approaches neither consider an existing code base, measurement-based testing techniques nor do they integrate decision support mechanisms or support the developer for implementing a solution at the code level.

In light of these observations, we are developing the Vergil approach (named after the ancient Roman poet Publius Vergilius Maro, Dante's guide through the inferno in The Divine Comedy [10]) that guides developers from a performance or scalability issue to solutions, by providing hypotheses about what to change, evaluating the changes in the context of the particular application and ranking the solutions to support developers in making a decision. Solution alternatives are provided as ordered lists of work activities sketching the implementation of the solutions for developers. Additionally, in order to support developers in making a decision on which solution to implement when different alternatives exist, it is necessary that developers estimate the effort to implement a certain solution. Therefore, developers often expect concrete work activities when asked for estimating the necessary effort. Vergil targets the development and maintenance phase of the software systems life cycle when an executable application implementation is available.

The core idea of Vergil is a process to identify and evaluate solutions to existing performance problems in a given application context, and to rank and recommend the most suitable of such solutions to the developers together with a description on how to implement them. The conceptual foundation is the formalization of performance expert knowledge into hypotheses about what to change and when. The goal of the approach is twofold: to make expert knowledge and methods for performance problem solution easily available to developers (who are not necessarily experts in performance engineering) and to guide them through the solution process with automation and tools [11].

In this paper, we introduce the overall concept of Vergil. The remainder of the paper provides an overview of Vergil's overall process, individual process activities, involved artifacts and how process activities and artifacts are connected.

 ISBN: 978-1-61208-367-4

In summary, we provide the following contributions in this paper:

1) We introduce Vergil's process, for exploring, evaluating and ranking of hypotheses to determine solutions of recurring performance and scalability problems.
2) We formalize performance problems as symptom traces in applications, and solutions as change hypotheses.

The remainder of the paper is structured as follows: In Section II, we introduce the process, its activities, and the artifacts of Vergil. In Section IV, we demonstrate the solution of a software bottleneck within a Java EE application through result caching. We present and discuss the related work in Section V and finally, conclude the paper in Section VI, also outlining our plan for future work.

II. The Vergil Approach

The main goal of Vergil is the provisioning of solutions (e.g., to split an interface or to move functionality to a certain component) to developers for solving performance and scalability problems. Vergil combines the strengths of a systematic process and the consideration of cost factors and constraints of model-based performance improvement approaches, for example [5][8][6], and extends them with the introduction of measurement-based performance problem solutions at the code level by means of monitoring-driven testing techniques, decision support mechanisms for selecting the most appropriate solution and work plans sketching the implementation of the solution.

There are two roles involved in Vergil as shown in Figure 1: Performance experts, who provide their knowledge about how to solve performance problems, and users (e.g., developers or other stakeholders; henceforth referred to as developers) who use Vergil to solve performance problems. The knowledge of performance experts about how to change a system is formalized in rules (henceforth called *Change Hypotheses*). Knowledge about how a change can propagate and impact other parts of the application is formalized in *Propagation Rules*. In each use case, developers provide the information about the problem context. They provide the *Performance Problem Model* by means of specifying the symptoms (e.g., high response times, high CPU utilization, or high memory utilization) and where they observe the symptoms in the application. They also provide the *Source Code* of the application, a *Test Environment* where the application can be deployed and the running application can be monitored during the execution of load tests, the *Performance Requirements* of the application as well as the willingness to change certain parts of the application, and constraints as *Developer's Preferences*.

Vergil uses all artifacts to test the applicability of *Change Hypotheses* and to evaluate which changes are leading to a performance improvement either with measurements and/or performance models. Vergil discards solutions that are not conforming with the *Developer's Preferences*. For each determined solution, Vergil derives a *Work Plan* with activities sketching the implementation of the solution. Developers estimate the implementation effort of each work plan. Vergil ranks the solutions based on all the collected information throughout

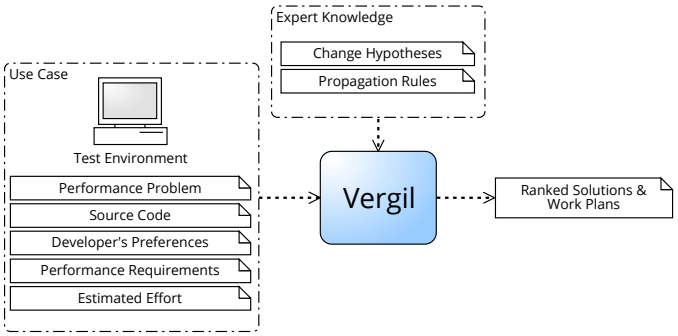


Figure 1: Vergil Overview.

the process and presents the ranked list as feedback to the developer. Developers can then discuss the solution proposals and implement the selected solution with the help of the *Work Plan*.

The process consists of four major activities as shown in the BPMN diagram [12] of Figure 2. In the context of this paper, we are focusing only on the overall concept of Vergil. The details of the activities *Propagate Work Activities* and *Estimate Effort of Work Plans* are described in [13].

A. Extract Models

The process starts with the *Extract Models* activity [14] that takes the source code of the application as input. The source code is parsed into the Source Code Model (SCM), for example with the Java Model Parser and Printer (JaMoPP) [15] for Java source code. An architecture performance model (APM) is extracted from the source code (in the context of this paper from Java) or when such a model already exists, it is imported. In the context of this paper, the APM is a Palladio Component Model (PCM) [16]. PCM is a software architecture simulation approach to analyze software at the model level for performance bottlenecks and scalability issues. It enables software architects to test and compare various design alternatives without the need to fully implement the application or buying expensive execution environments. PCM has already been used to detect and solve performance problems [7]. The PCM is created from the source code using the Software Model Extractor (SoMoX) [17]. The APM provides an architectural view of the application and is used to evaluate architectural change hypotheses in the remainder of the process. During the APM extraction, the Correspondence Model (COM) is build that links APM and SCM model elements, for example interfaces. A correspondence expresses the equality relation of two model elements in different meta-model instances. The SCM, APM, and COM are forwarded to the *Explore Change Hypotheses* sub-process.

B. Explore Change Hypotheses

The sub-process consists of the four activities *Test Change Hypotheses*, *Propagate Work Activities*, *Evaluate Work Activities*, and *Extract Work Plans*. Before we are going into the details of each activity in the remainder of this section, we introduce the performance problem model and our concept of change hypotheses. Change hypotheses provide the knowledge about what can be changed to solve a performance problem. The hypotheses are an important cornerstone in Vergil and are rules expressing what to change and when.

 ISBN: 978-1-61208-367-4

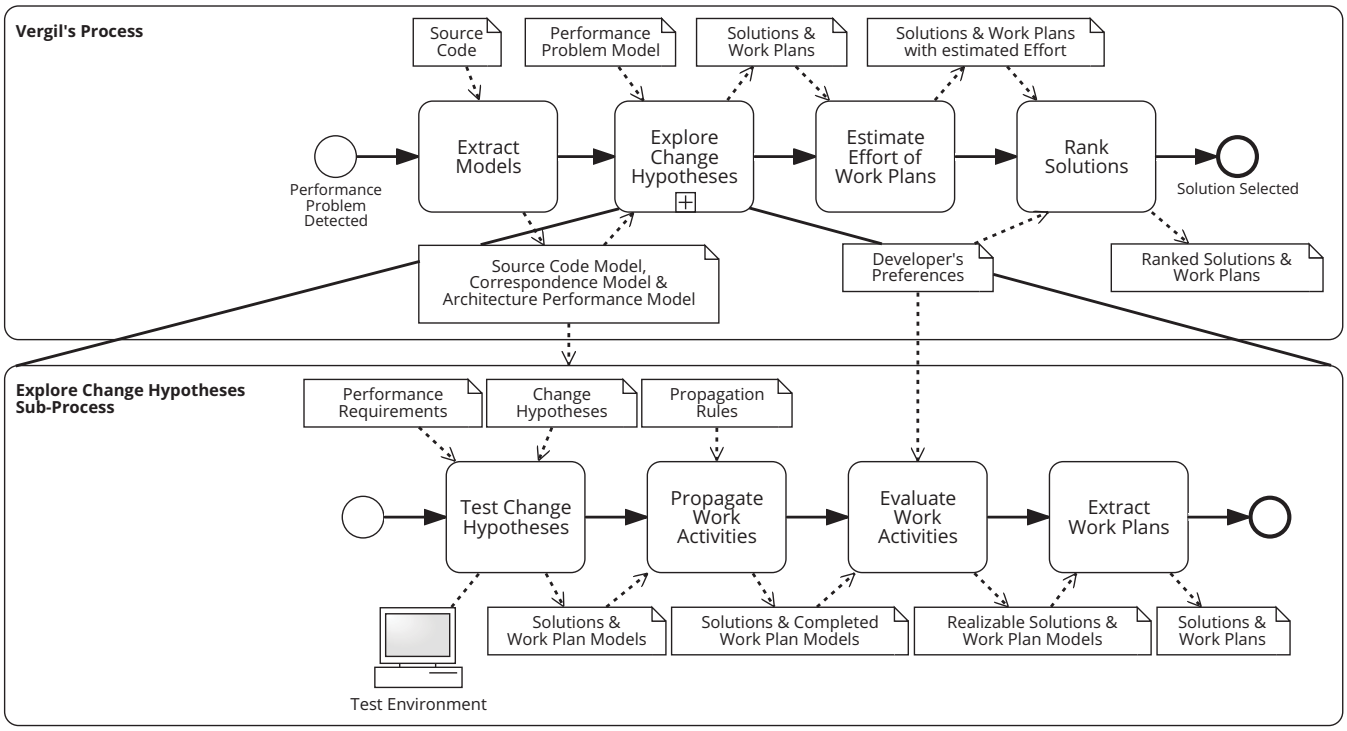


Figure 2: Vergil Process Overview.

Definition 1: A performance problem is a symptom trace through the application. It is formalized through a model of its root cause(s) expressed by its symptom(s), the workload specification [18] including the usage profile, and location(s) inside the application as shown in Figure 3. The resulting model is henceforth called *Performance Problem Model (PPM)*. A performance problem can have any number of other performance problems as cause expressed through the *causedBy* relation. One or more symptoms belong to a performance problem. A symptom can be among others: high CPU utilization, high response time, high memory utilization or high network utilization. A location is the referenced element of the SCM like a class, method, or statement where the symptoms are observed. The workload specification describes the workload (e.g., the number of users and their think time in a closed workload scenario) and the usage profile under which the symptom can be observed. The workload specification is formalized as a finite state machine and probabilistic usage behavior by means of Markov chains [18].

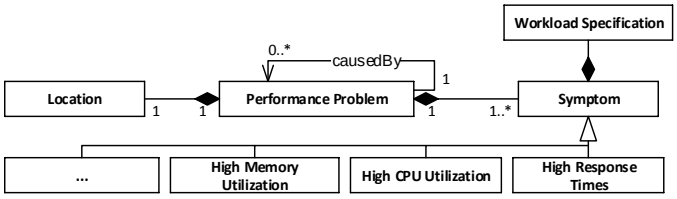


Figure 3: Performance Problem meta-model.

The PPM can be automatically extracted from a tool such as DynamicSpotter [19] or instantiated manually by the developer and is given as input to the process.

Definition 2: A change hypothesis h consists of a set of preconditions that must be fulfilled in order to be applicable in the problem context, a set of transformation rules that apply the changes of the hypothesis to the application on the defined level of abstraction (e.g., APM, SCM, etc.), a set of postconditions that test if the expected effect has taken place, and a work plan model template for creating the initial work plan model as shown in the meta-model in Figure 4. The conditions can test structural or behavioural properties of the application. A condition can consist of any number of structural (on the SCM, PPM and APM model) and behavioral (on measurement or prediction results) conditions testing static and dynamic requirements of the hypothesis. The conditions are rules expressed as logical predicates in first-order logic. First-order logic has already been used before in literature to formalize performance antipatterns [20]. The formalization of the changes depends on the level of abstraction. For example, in the case of APM and SCM (basically Java source code), graph rewriting rules (in place transformations) are used to transform the models. In the case of modifying parameter values in configuration files, simple text replacement rules are used.

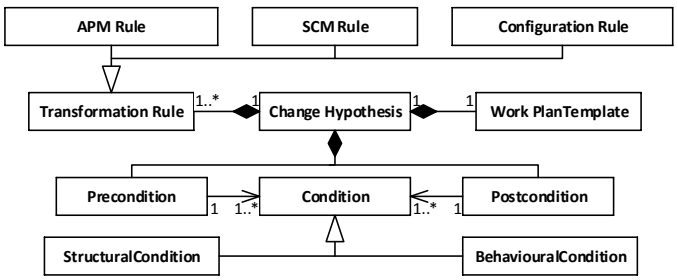


Figure 4: Change Hypothesis meta-model.

In the following, we provide an example of a hypothesis that caching the results of calling a method can improve performance:

The behavioural precondition of the hypothesis that caching the results of calling a method can improve performance ensures that the method m producing the results is deterministic. Deterministic means that for each method input $i \in \mathbb{I}$ and for all method calls c_m of method m there exists only one result $r \in \mathbb{R}$ in the set of results so that $c_m(i) = r$. The formalization of the precondition as one basic predicate *BP* is as follows:

$$\forall i \in \mathbb{I}, \forall c_m \in methodCalls(m), \exists! r \in \mathbb{R} : c_m(i) = r \quad (1)$$

where m denotes the method whose results shall be cached.

The structural precondition of the hypothesis matches a pattern in the SCM and PPM where a method is referenced from a performance problem with the "High Response Time" symptom and where the method implements an interface method. The structural precondition is formalized through the *BP* where m denotes the method (referenced by the performance problem) and m' denotes the interface method in the set of all methods:

$$\exists m, \exists m' \in Methods \subset SCM : \exists p \in PPM : impl(m, m') \wedge ref(p, m) \wedge has(p, HighRespTimes) \quad (2)$$

The postcondition of the hypothesis ensures that the number of method calls of m has decreased after the changes of the hypothesis have been applied.

Taking the PCM as APM, the transformation rule of the hypothesis targets the PCM instance of the application. The performance-relevant behaviour of a method in PCM is modelled through a Service Effect Specification (SEFF) [16] (basically a series of actions) that can contain among others *BranchAction*, *InternalAction*, and *ExternalCallAction* elements. A *BranchAction* models a branch and can take the probabilities for each transition. An *ExternalCallAction* models the call to a method of another component. On an abstract level, caching results means that there is a probability P that the result is in the cache. This can be modeled in its simplest form by means of putting the *ExternalCallAction* EA, calling the method whose results shall be cached, inside a *BranchAction* that has the probabilities P for the cache hit transition and $1 - P$ for the cache miss transition (modeled through a *ProbabilisticBranchTransition*). Therefore, the rule simply wraps EA in the SEFF into a *BranchAction*.

The work plan model (WPM) template of the hypothesis is a blue print to create the initial work activities. A hypothesis knows the work activities resulting from the changes but not the possible side-effects. For example, a hypothesis to split an interface knows the work activity "Split". The work plan meta-model is introduced in the context of the *Extract Work Plans* activity in the remainder of the paper.

This concludes the hypothesis example. In the following, we describe the activities of the sub-process:

1) Test Change Hypotheses: The *Explore Change Hypotheses* sub-process starts with the *Test Change Hypotheses* activity that takes the change hypotheses H, the test environment, the performance requirements and the models as input. In this activity, the applicability of change hypotheses is tested, and the effect of the hypotheses' encapsulated changes is evaluated to build solutions. Jing Xu [8], Mauro Drago [21] and Diaz-Pace et al. [22] already considered performance evaluation of changes. The exploration algorithm selects sets of change hypotheses with fulfilled precondition and evaluates their effect through instantiating the changes in the context of the particular application and on the hypothesis' level of abstraction (e.g., architecture performance model, source code, or configuration file) and evaluates the performance. Vergil only considers changes that can be applied automatically. To give an example, two approaches for automated model refactoring for solving performance problems are presented in [23] and [8].

The set of change hypotheses H is the input for the *EXPLORE* procedure of the exploration algorithm as shown in Figure 5 (line 5). The current algorithm uses backtracking (as suggested by Arcelli and Cortellessa [9] and used in [24]) to find solutions that fulfill the performance requirements. The basic backtracking algorithm loops over all $h \in H \setminus H'$ and evaluates the precondition of h after the changes of the hypotheses in H' have been applied (line 6-7). If h_{PreCon} evaluates to $true$, the hypothesis h is evaluated. The postcondition of h is evaluated on the returned evaluation $result$ (line 8). When $h_{PostCon}$ evaluates to $true$ then h is added to H' (line 9-10). When the changes of a hypothesis are applied, the impacted elements of the application are identified as well as how they are impacted and also returned in $result$. The information is used to build the work plan models and its initial work activities based on the template. When $h_{PostCon}$ is not fulfilled, then the loop continues with the next hypothesis $h \in H \setminus H'$ (line 17). When the postcondition and the performance requirements are fulfilled, the hypotheses composition H' is added as solution to $solutions$ (line 12).

The performance requirements are expressed as upper bound of a performance metric. For example, the performance metric can be the response time of a method, the CPU, memory and/or network utilization of a server the application is running on. The performance evaluation and the postcondition also ensure that the changes do not lead to a performance degradation [9]. If the postcondition is fulfilled, but the performance requirements are not fulfilled, the procedure *EXPLORE* calls itself recursively with hypotheses composition H' and the set of hypotheses H (line 14). The result of the algorithm is the set $solutions$ that consists of sets of change hypotheses. Mathematically, the basic algorithm can miss solutions as it does not check all possible combinations of hypotheses (for practical reasons). However, a variation of exploration algorithm to test all possible compositions has to neglect the pre- and postcondition tests.

The performance improvement in Line 8 is either estimated by means of prediction through APM or determined through measurement-driven testing techniques on the System Under Test (SUT) as shown in Figure 6. The SUT consists of the deployed application and the *Test Environment (TE)*. The TE is a testing and monitoring environment where the application can be deployed and executed. Part of the TE is a load generator (e.g., HP LoadRunner [25], Apache JMeter [26]) and a representative load test for the application that is used to simulate users using the application. The load test itself consists of: a usage profile (how the application is actually used by its users), and the number of users to simulate and their think time (in a closed workload scenario). The usage

```
1: Set H ← Change Hypotheses
2: Set H' ← ∅
3: Set solutions ← ∅
4:
5: procedure EXPLORE(Set H', Set H)
6:     for all h ∈ H \ H' do
7:         if evaluate(h_PreCon, H') then
8:             result ← evaluate(h, H')
9:             if evaluate(h_PostCon, H') then
10:                H' ← H' ∪ {h}
11:                if solved(result) then
12:                    solutions ← solutions ∪ (H', result)
13:                else
14:                    explore(H', H)
15:                end if
16:            else
17:                Continue
18:            end if
19:        end if
20:    end for
21: end procedure
```

Figure 5: Exploration of Change Hypotheses.

profile is often a probabilistic behaviour. Given a currently visited web site, a user visits another web site or selects a certain element with a certain probability. The probabilistic usage profile and intensity-varying workload is specified in two types of models. A finite state machine specifies the possible interactions with the Web-based software system. Based on the finite state machine, the probabilistic usage is specified in corresponding user behavior models by means of Markov chains [18]. Markov4JMeter [27] implements such an approach for probabilistic workload generation by extending the workload generation tool JMeter. The probabilities can be determined from real-user monitoring of a deployed application running in production or manually through the expected usage of the application when no deployed application is available. In the latter case, a common, non-probabilistic usage profile is used.

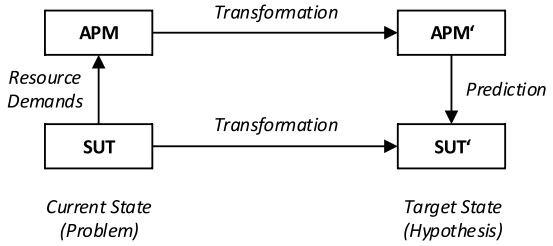


Figure 6: Evaluation Mean Alternatives.

For measurement-based experiments by means of monitoring-driven testing techniques on the SUT, our Adaptable Instrumentation and Monitoring (AIM) agents are deployed on the servers of the TE to instrument Java bytecode to monitor the application under load and to sample the resource utilization (CPU, network, etc.) of the servers. AIM provides means to automate the adaptation of instrumentation instructions. In experiment-based performance engineering, this feature can be utilized to automate a series of experiments to make manual interventions between individual experiments unnecessary. AIM specifies an extendable language to describe a desired instrumentation and monitoring state on an abstract level. It parses instances of the instrumentation description model and realizes instrumentation and monitoring instructions utilizing bytecode instrumentation, sampling and interception of the underlying Java Virtual Machine (JVM). A separate publication on AIM is in progress.

When the effect of changes is evaluated, the evaluation starts in *Current State* (cf. Figure 6) where a series of reference measurements S_0 is obtained. In the case of a $SUT \rightarrow SUT'$ evaluation, the measurements are obtained by means of monitoring-driven testing techniques with AIM and the execution of a workload. The source code transformation rules are applied to the SCM. The transformed SCM is used to transform the source code, for example in the case of Java, with JaMoPP. Configuration transformation rules are applied to the configuration files. Component configurations specified in the source code are treated as implementation transformation rules. The application of the transformation rules creates the SUT'. The series of evaluation measurements S_1 from the SUT' are then obtained analogous to S_0.

In the case of a $APM \rightarrow APM'$ evaluation, the resource demands for the calibration of the APM instance are obtained from the SUT. To determine the resource demands, the corresponding source code regions are instrumented with AIM to derive cumulative distribution functions $CDFs$ through experiment-based measurements with the SUT. The determined resource demands are inserted into the APM instance. The series of reference measurements S_0 is obtained through simulating the calibrated APM instance. The transformation rules are applied to the APM instance to create the APM'. In the *Target State*, APM' is simulated to derive the evaluation measurements S_1, as a prediction for the measurements expected from SUT'.

In both evaluation scenarios, the estimated performance improvement is determined as the difference $S_0 - S_1$. After the *Target State* is reached and S_1 is obtained, the changes are reverted (cf. Figure 5, line 8). The solutions and the corresponding WPMs fulfilling the requirements are forwarded to the *Propagate Work Activities* activity.

2) Propagate Work Activities: In this activity, the WPMs are completed by identifying all impacted elements of a solution and determining for any impacted element the required work activity. Vergil uses impact propagation rules to accomplish this task. The directly affected elements are identified during the instantiation of a change hypothesis (because it is applied to these elements). The WPM template of the change hypothesis provides the initial set of work activities. Changes can ripple through the application impacting other elements that are in a relationship (side-effects). To complete the WPMs, the side-effects and their work activities are determined through impact propagation rules [28]. Impact rules know if and how a work activity propagates itself to other elements, for example, when an interface in SCM is referenced from a "Split" work activity, then the rule knows that a class implementing that interface has to be splitted too. A side-effect can also be that new tests have to be added when a new interface is going to be created. Another example for such a follow-up activity is the redeployment of a component if the implementation will undergo changes. Rules are also used to conclude follow-up activities. The completed WPMs with the propagated impact and the corresponding solutions are forwarded to the *Evaluate Work Activities* activity.

 ISBN: 978-1-61208-367-4

3) Evaluate Work Activities: In the *Evaluate Work Activities* activity, the work activities are validated against the *Developer's Preferences*. Their referenced elements are tested if they can be changed. Developers express their willingness or unwillingness to execute a certain type of change on a grading scale. They can also express what cannot be changed, such as legacy parts of the application or the database. Arcelli and Cortellessa [9] already raised the concern that cost factors and constraints (e.g., the database cannot be changed) have to be taken into account when proposing solutions. Solutions whose WPMs contain unchangeable elements are discarded, removed from the set of solutions and the corresponding WPM is deleted. In the case of Java programs, Vergil takes unchangeable elements into account through the specification of the full qualified name or namespaces by the developers. For example, if a work activity references an element in the APM then an architecture impact is concluded. If the architecture is unchangeable, then the solution is discarded. The set of remaining solutions and WPMs are forwarded to the *Extract Work Plans* activity.

4) Extract Work Plans: In the *Extract Work Plans* activity, a list-based representation of the WPMs for the developers is extracted. The list of work activities sketches the implementation of the corresponding solution for developers. It also serves as foundation for the *Estimate Effort of Work Plans* activity. The list structure is determined through the *refinedBy* and *dependsOn* relations between work activities in the WPM. The *refinedBy* relation expresses the parent-child relationship of work activities whereas the *dependsOn* relation expresses the order of work activities.

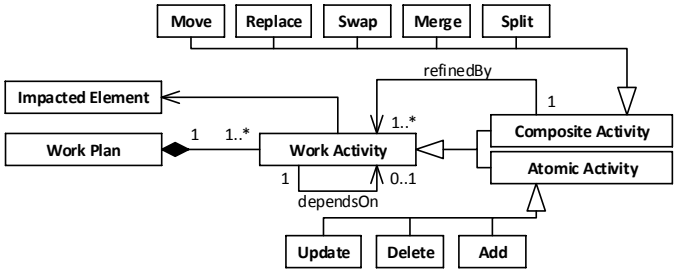


Figure 7: Work Plan meta-model.

Definition 3: A work plan is an ordered set of work activities. A work activity can be atomic such as add, delete, or update an element like a class, interface, and method or composite such as split, move, merge, swap, or replace elements [29]. A composite activity can be composed of other composite and atomic activities and is broken down until it is expressed through atomic activities. Refinement rules are used to break composite activities in the WPM down into atomic activities.

Work plans are not prescribing how the solution has to be concretely realized in the application. Jing Xu already motivated in [8] that the solution suggests what should be changed in an abstract way, but not how to do it concretely because there can be a host of ways. Vergil accomplishes this by modelling only abstract work activities sketching the implementation of a solution. The work plan can also, if necessary, list follow-up activities such as redeployment work activities and testing activities. The *Explore Change Hypotheses* sub-process ends after the extraction of the work plans is completed and forwards them together with the solutions to the *Estimate Effort of Work Plans* activity.

C. Estimate Effort of Work Plans

In the *Estimate Effort of Work Plans* activity, the effort for any work plan application is estimated by developers. This is a manual task done by the developers themselves because the effort can vary between individual developers depending on their knowledge, experience and practice. Vergil accepts the effort as unit less quantities for all atomic work activities. This leaves the decision of the concrete unit of measurement by the developers. Chosen once (in the current execution of the process), the unit of measurement has to remain the same for all work activities and work plans. The effort can be estimated, for example, in person (-hours, -days, or -months) [8]. The total effort estimation for a work plan is the computed sum of the unit less quantities of each atomic work activity. The consideration of the estimated effort takes the costs of solution alternatives into account [8], [9]. The solutions and the work plans with estimated effort are forwarded to the *Rank Solutions* activity.

D. Rank Solutions

In the *Rank Solutions* activity, the solutions are ranked through the rating of a multi-criteria decision analysis. The rating takes costs and constraints into account to support developers in deciding on an appropriate solution when a variety of choices exists [8][9]. The rating is done similar to [30] with a combination of the Analytic Hierarchy Process (AHP) [31] and the Simple Multi-Attribute Rating Technique (SMART) [32] taking the performance impact, cost factors, constraints and the developer's preferences into account. In the first step, AHP is used to obtain the priorities of the criteria. In pairwise comparisons, the developer judges the importance based on a fundamental scale of absolute numbers [31]. The priorities are given as input to SMART. SMART is a method of the multi-attribute utility theory. In contrast to the AHP where decision-making is done through pair-by-pair comparison of alternatives requiring human intervention, SMART ranks the solution alternatives based on the information already collected throughout the process using the given priorities (henceforth referred to as weights).

SMART uses a decision table consisting of m criteria $C_1, C_2, \ldots, C_m$ as rows and n solution alternatives $A_1, A_2, \ldots, A_n$ as columns. The cells contain the value of the alternative with respect to the criteria. Each criteria has an assigned weight w_i as dimensionless, normalized number originating from the developer's judgements (e.g., the importance of performance improvement, effort, or the willingness to change the architecture, etc.). For all alternatives, SMART computes the rating x_j of alternative A_j as follows:

$$x_j = \frac{\sum_{i=1}^{m}(w_i\ a_{ij})}{\sum_{i=1}^{m} w_i}, \quad j = 1, 2, \ldots, n \tag{3}$$

where a_{ij} is the normalized value of criteria C_i and alternative A_j.

The list of solutions is sorted descending according to the computed SMART ratings $x_1, x_2, \ldots, x_n$. The solution with the highest SMART rating in the list is placed on top.

 ISBN: 978-1-61208-367-4

Developers are then able to review and discuss the proposed solutions based on the work plans, the impacted elements—and how they are actually impacted, the costs, and the estimated performance improvement and to select a solution they are willing to implement. The selected solution and its work plan are the final result of the process.

III. DISCUSSION

In this section, we clarify the current implementation status of Vergil's framework, the dependency on component-based software architectures and programming languages, and different categories of refactoring changes.

A. Automation of process activities

Currently, there are implementation prototypes for the two activities *Rank Solutions* and *Propagate Work Activities*. The activities of the *Explore Change Hypotheses* sub-process are intended to be automated in the near future. The *Estimate Effort of Work Plans* activity is not automated. However, Vergil still supports the developer by providing work plans sketching the implementation steps. In our next steps, we are designing the architecture of Vergil's framework based on the feedback we have received. Our goal is to design an architecture that allows tailoring the process to the specific needs of a use case. To give an example, in a certain use cause, it might be infeasible to estimate the implementation effort for each solution alternative. Instead, it is only feasible to estimate the implementation effort of the top-k solution candidates. Therefore, the solution alternatives must be ranked based on the criteria (neglecting the implementation effort) to identify the top-k solution candidates. In such a scenario, the rank solutions activity must occur twice in the process: (1) before the *Estimate Effort of Work Plans* activity, and (2) thereafter considering only the top-k candidates. In another use case, the developer may want to have all solution proposals in the ranking regardless of their conformity with the *Developer's Preferences*. In this case, the *Evaluate Work Activities* must be skipped.

B. Extension of the framework

Conceptually, Vergil is designed to be applicable to applications following component-based architecture and object-oriented design principles. The implementation of Vergil in the context of our research focuses on the Java programming language and the Palladio Component Model, which are both established means in industrial practice. We designed Vergil's framework to use exchangeable plugins to be able to support different programming languages and technologies. We specify and provide interfaces to implement plugins, e.g., to support the C# programming language. However, to make Vergil support other languages, there are certain key aspects that must be considered: programming language and technology specific knowledge encapsulated in Vergil's artifacts must be changed, extended, or developed to work with different languages and technologies.

C. Proposal of non-automatic evaluable solutions

In general, the changes of a change hypothesis can be assigned to one of the following three categories: (A) automatically executable, (S) semi-automatically executable, and (M) manually executable. Each category determines the ability to apply the changes of a change hypothesis automatically and the demand of human intervention, in order to evaluate the performance improvement of a change hypotheses (or a solution in general). The execution of refactorings in a work plan can also be categorized into (A), (S), and (M). As a result, we distinguish between nine possible categories of solutions described by the tuple:

$$Solution\ Category = Cat(DoA_{Ev}, DoA_{Ex}) \qquad (4)$$

where DoA_{Ev} determines the Degree of Automation (DoA) for evaluating the changes of a change hypothesis (the application of changes to the system respectively) and DoA_{Ex} determines the degree of automation for the execution of a work plan.

Categories (A, A), (A, S), and (A, M) are expected to require no human intervention to evaluate the performance improvement of a change hypotheses. Category (S, S) and (M, M), on the other hand, require human intervention. The category is often determined through the complexity of the refactoring. For example, simple refactorings like changing annotations are categorized as automatically executable. Refactorings categorized as semi-automatically or manually executable require the implementation of refactorings prior to their evaluation which is infeasible in most cases (cost vs. benefit trade-off). In order to avoid the implementation of changes just to evaluate the performance improvement, Vergil can still provide solution proposals in terms of work plans but without evaluating the performance improvement accompanied to the risk of introducing a performance degradation. Nevertheless, the proposal of such solutions can still be valuable for developers.

IV. MEDIASTORE EXAMPLE

In this section, we provide an outlook on the validation of Vergil. We present an excerpt of the *Test Change Hypotheses* activity by evaluating the change hypothesis given as example in Section II-B to cache the results of a method with the high response time symptom in a $APM \rightarrow APM'$ evaluation scenario. The example is structured as follows: In the current state, measurement-driven experimenting techniques are used to determine the resource demands from the SUT and to calibrate the *APM*. The calibrated *APM* is simulated to obtain the series of reference measurements S_0. Then, the hypothesis' changes are applied to transform the APM into APM'. The APM' is simulated to obtain the series of evaluation measurements S_1 (cf. Figure 6 and Section II-B1). To present preliminary results, we also show the response time measurements before and after implementing the changes in Figure 8.

We use a MediaStore [16] application as a simple use case example accessing the database and processing the data. The MediaStore allows its users to upload and store audio files as well as to download audio files encoded in a less or equal audio bit rate compared to the uploaded one. The application is implemented in Java EE and is deployed in a GlassFish 4 application server with Derby 10.10 as the database management system. The application server and the database management system are located on separate nodes. A short overview of the most relevant components for the example is shown in Figure 9 as excerpt from the PCM model. Only features relevant for the example are shown here and other

 ISBN: 978-1-61208-367-4

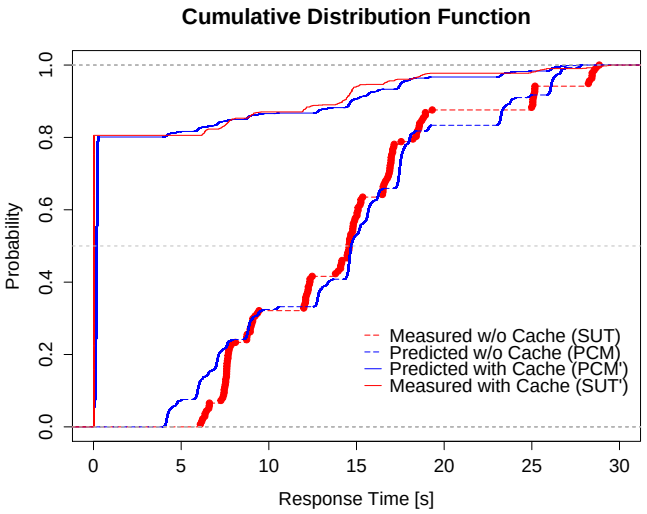


Figure 8: Measured and predicted response times.

PCM features can be found in [16]. We use the PCM model as APM in the example. The PCM model shows the resource container, on which the `WebGUIBean`, `MediaStoreBean`, `AudioAdapterBean`, and `EncoderBean` components are deployed. The resource container corresponds to the node in the *Test Environment* on which the MediaStore is deployed for measurement-based experiments. The SEFF models the performance-relevant behaviour of the `MediaStoreBean`'s download method and consists of the external call action to fetch an audio file from the database and the external call action to encode the audio file in a specified audio bit rate.

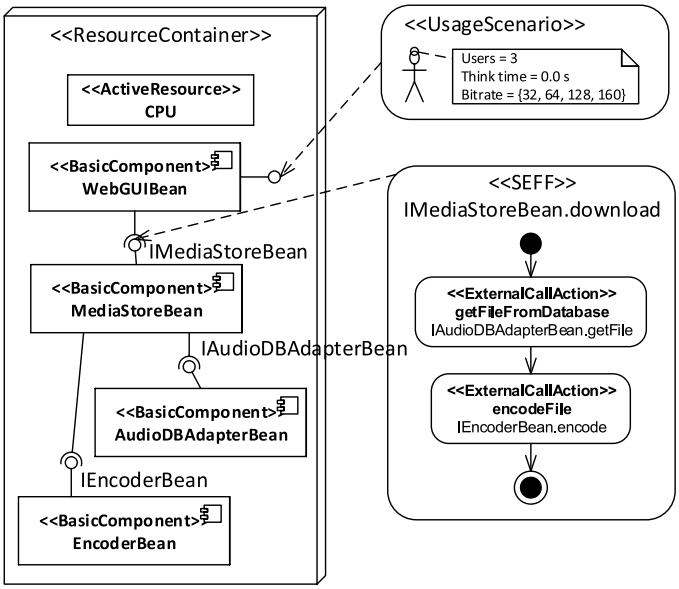


Figure 9: MediaStore PCM model excerpt.

We consider a scenario where multiple users download an audio file $\alpha \in AudioFiles, |AudioFiles| = 81$ randomly with a bit rate $\beta \in B = \{32, 64, 128, 160\}$ that is less compared to the uploaded bit rate of 190 kBit/s to force the re-encoding of α with bit rate β. Mathematically, the encoding function is defined as follows:

$$encode(\alpha, \beta) = \alpha' \quad (5)$$

where α' is the re-encoded audio file α in the desired bit rate β. The simulated usage profile is as follows: Users login, select the desired audio file α and bit rate β randomly following a uniform distribution, download the re-encoded audio file a', and logout. We simulate three power users with zero think time who execute the usage profile in a closed workload scenario using HP LoadRunner.

We instrument the `WebGUIBean`'s download method with our AIM agent and monitor the response time of the method. Therefore, the agent adds code statements at the beginning and the end of the method's body at the byte code level. The instrumentation (manipulating the byte code) is already fully automated. The added byte code instructions measure the time it takes to execute the method. In the monitoring results, shown as cumulative distribution function in Figure 8 (as dashed red line), we observe a measured median response time $\bar{r}_{mea} = 14.29s$ of the SUT in the applied workload and usage scenario. This is high in our considered scenario. The high response times are caused by the re-encoding of α.

We use the hypothesis (given as example in Section II-B) that caching the results of calling the `encode` method can improve performance. The `encode` method (as formalized in Equation 5) fulfills the precondition as it returns for the same input tuple (α, β) the same result α'. The size of an object cache is often specified by the number of elements that can be added to the cache before eviction takes place. In the case of a data access profile following a uniform distribution like in this example, the cache hit probability P only depends on the size of the cache and the total number of elements. For example, to achieve a hit probability $P = 0.8$, the cache size can be determined as follows:

$$\lceil |AudioFiles| * |B| * P \rceil = \lceil 81 * 4 * 0.8 \rceil = 260 \quad (6)$$

where the result is rounded to the next integer. In general, the size of the cache can be limited by the amount of memory that is available for caching objects. For the $APM \rightarrow APM'$ evaluation of the changes, we use the PCM model as shown in Figure 9, as APM. We extract the resource demands for the internal actions (not shown in Figure 9) in the SEFF of `IAudioDBAdapter.getFile` and `IEncoder.encode` with measurement-driven experiments on the SUT. The extraction is done (semi-) automatically. We are currently working on the full automation of resource demand extraction for PCM models with measurement-based experiments in the context of our publication about AIM. We calibrate the PCM model with the determined resource demands and simulate the usage profile and workload to obtain the series of reference measurements S_0. In S_0, also shown as cumulative distribution function in Figure 8 (as dashed blue line), we observe a predicted median response time $\bar{r}_{pre} = 14.39s$ in the current state of the APM.

In order to evaluate the hypothesis, we manually transform the SEFF `IMediaStoreBean.download` as shown in Figure 10. How the transformation can be automated is shown in literature [7][23][33]. We introduce a *BranchAction* and two *ProbabilisticBranchTransitions (PBT)* to simulate the cache. We assign the hit probability $P = 0.8$ to the *CacheHit* PBT and the miss probability $1-P = 0.2$ to the *CacheMiss* PBT. We assume the cache access time to be negligible, based on our practical experience (fetching an α' from the cache takes on average $0.02\mu s$ in the SUT' with the implemented caching solution).

 ISBN: 978-1-61208-367-4

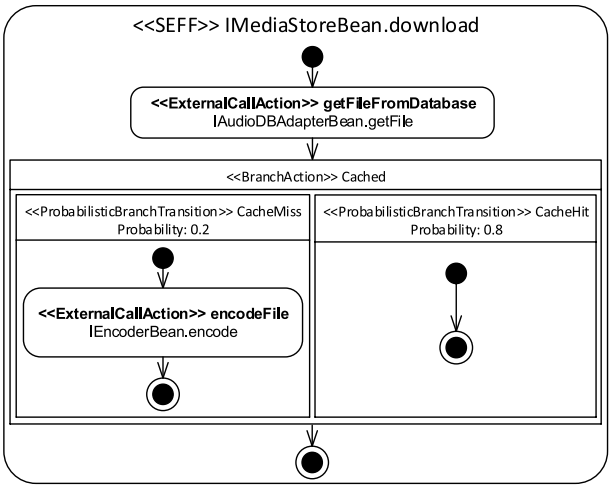


Figure 10: SEFF in target state of APM' with simulated cache.

The simulation results S_1 (denoted as blue line) for PCM' are shown in Figure 8 as cumulative distribution function. The simulation predicts a median response time $\bar{r}'_{pred} = 2.95s$. Based on the evaluation results, a performance improvement of 487% is estimated for the changes of the hypothesis.

To validate the simulation results, we also executed the $SUT \rightarrow SUT'$ evaluation. We implemented the cache as an Enterprise Java Bean with the help of Google's Guava libraries [34]. In the implementation of the `MediaStoreBean.download` method, the cache is checked first for the tuple (α, β). When the audio file cannot be obtained from the cache, α is fetched from the database and re-encoded with bit rate β. The resulting α' is added to the cache. We set the cache size to 260 objects and repeated the load test. The initial warm-up of the cache is done during the ramp-up phase of the load test. In the monitoring results (as shown in Figure 8 denoted as red line), we observe the measured median response time $\bar{r}'_{mea} = 2.71s$. The measured response times show a performance improvement of 527%.

V. RELATED WORK

The comprehensiveness of Vergil's process leads to a broad area of related fields of research. In the following, we cite only the most important and most relevant approaches to performance problem solutions due to space constraints. The interested reader may refer to [21][35] for more details about meta-heuristic approaches and generic design space exploration approaches. We categorize related approaches into model-based and measurement-based performance solution approaches.

A. Model-based Performance Solution

In [35], Cortellessa et al. present a model-based approach to automatically detect and solve performance antipatterns. Their approach targets the early design phase and the suggestion of architectural design changes to overcome performance problems. The goal of their proposed process is to modify a software system model to produce a new model without the performance problems of the former one [35]. They formalize antipatterns (often defined in natural language) as logical predicates in first-order logics [20][36]. Arcelli et al. present the automation of the model refactoring to improve the performance by applying model differences based on a Role-based Modeling Language [6][23]. Their approach suggests developers how to refactor models in order to remove problems. In the context of the approach of Cortellessa et al., Trubiani and Koziolek present the detection and rule-based solution of performance problems in Palladio Component Models in [7]. In [9], Arcelli and Cortellessa raise the need to take cost factors and constraints into account when a variety of solution choices exists and to integrate decision support mechanisms to support designers in selecting the most appropriate solution(s). In [8], Jing Xu presents a rule-based approach to detect and solve performance bottlenecks and long-path performance problems based on performance models. Models are modified with the help of the rules in ways that can be converted to design changes, which are then done manually. The costs for changing the design (carried out manually) is taken into consideration and can discourage rules from selecting changes, on a cost-effectiveness basis and for practical reasons. The proposed design changes describe what should be changed, and in what way, but not how to do it. The search is an iterative process. In each round, multiple alternatives can be created and the performance improvement is evaluated. When multiple design change branches are obtained at the end of each round, the performance improvement and weight of each branch is listed and ranked. Solutions are provided at the performance model level. Designers have to transfer the solutions from the performance model level to the design model level. In [37], Martens et al. propose an approach for automated performance improvement of component-based software systems based on meta-heuristic search techniques and rules applied to Palladio Component Models to find solutions for performance problems. In [21], Mauro Drago automates the detection-solution loop to automatically generate and propose design alternatives as feedback to the designer to improve non-functional properties of a software design. Quality-driven transformations are used to generate alternatives. Queuing Networks are used with estimated service demands to predict non-functional properties of each alternative. Diaz-Pace et al. [22] propose a framework to assist the software architect in the design of software architectures meeting quality requirements. Rules are used to change the design of the system. Currently, only rules to improve modifiability are supported that are applied to a graph-based representation of the architecture. The modifiability is evaluated with change impact analysis to determine the cost of changes while the performance is predicted with a simple performance model.

Neither of the approaches presented above considers an existing code base and measurement-based performance problem solutions nor do they support the developer in implementing the solution with an ordered list of work activities. The detection of performance problems with measurements and/or performance models is not in focus of Vergil. Also, Vergil targets the software development and maintenance phase of a systems lifecycle, when an implementation of the application is available. Vergil calibrates performance models with resource demands obtained from the system under test providing a more representative evaluation.

B. Measurement-based Performance Solution

Currently, to the best of our knowledge, there is no measurement-based performance solution approach that considers a comprehensive process for performance problem solution. In [14], Trevor Parsons uses monitoring-based techniques to

 ISBN: 978-1-61208-367-4

extract a performance model of a Java EE application. The performance model is searched for detecting EJB-specific performance antipatterns. Problem solution is not part of the approach. To improve the deployment of components, Malek et al. introduce a framework [38] that guides the developer in the design of their solutions for component redeployment for large distributed systems. The goal is to find a deployment architecture that exhibits desirable system characteristics or satisfies a given set of constraints. They use runtime monitoring and consider quality of service (e.g., latency, availability). Aled Sage presents in [39] an approach for the observation-driven configuration of complex software systems. The author uses established statistical methods from manufacturing, called Taguchi Methods, and experiments to find configurations such as communication concurrency that meet the needs of various stakeholders. Lengauer and Mössenböck [40] propose the use of iterated local search methods to automatically compute application-specific Java garbage collector configurations. The selected configuration candidates are evaluated with monitoring-based techniques. The evaluation results are used to solve an objective function to determine the best configuration. In [41], Chen et al. use measurement-based experiments and source code changes in the context of object relational mapping only to prioritize the solution of performance problems based on the estimated performance improvement. However, their proposed approach does not consider performance problem solution.

Existing measurement-based approaches are focused mainly on a particular problem and its solution at the configuration level or at the architecture level. Neither of the approaches also considers the solution of performance problems at the code level nor do they provide a comprehensive process guiding the developer from a problem to a solution with work activities. Also, neither of the approaches consider cost factors and constraints for selecting the most appropriate solution when a variety of solution alternatives exist.

VI. Conclusion

Vergil guides developers from a detected performance or scalability problem to a solution. The proposed process explores hypotheses about what solutions can be applied to the software system, evaluates the performance improvement based on measurements and/or performance models, and ranks the solutions with respect to performance improvement, cost factors, constraints and the developer's preferences. The solutions are presented as an ordered list of work activities, sketching the implementation of the solution without prescribing to the developer how the solution is actually implemented. Strong concepts already used in existing model-based approaches are brought together and are extended with measurement-based performance problem solutions at the code level and the integration of decision support mechanisms to support the developer in selecting the most appropriate solution when a variety of choices exists. Vergil provides a comprehensive process for solving performance problems in the development and maintenance phase of an application's lifecycle where an implementation exists and where the solution of performance problems is known as expensive [42]. In this work, we presented the main idea, the details of the process and its activities as well as the formalization of performance problems and performance expert knowledge. Using an example, we presented promising preliminary results as a proof of concept for measurement-based performance problem solution and the calibration of performance models. We are currently working on the validation of the overall approach on a case study with a large open source e-commerce system. Additionally, we plan to conduct an empirical study with software performance consultants and developers.

Acknowledgment

The authors would like to thank Jonas Kunz and Sven Kohlhaas for their support that contributed to the paper. This work is supported by the German Research Foundation (DFG), grant RE 1674/6-1.

References

[1] C. Smith and L. Williams, "More new software performance antipatterns: Even more ways to shoot yourself in the foot," in CMG-CONFERENCE-, 2003, pp. 717–725.

[2] B. Dudney, S. Asbury, J. Krozak, and K. Wittkopf, J2EE antipatterns. Wiley, 2003.

[3] C. U. Smith, "Performance engineering of software systems," Addison-Wesley, vol. 1, 1990, p. 990.

[4] A. Georges, D. Buytaert, and L. Eeckhout, "Statistically rigorous java performance evaluation," ACM SIGPLAN Notices, vol. 42, no. 10, 2007, pp. 57–76.

[5] V. Cortellessa, A. Di Marco, R. Eramo, A. Pierantonio, and C. Trubiani, "Approaching the model-driven generation of feedback to remove software performance flaws," in Software Engineering and Advanced Applications, 2009. SEAA'09. 35th Euromicro Conference on. IEEE, 2009, pp. 162–169.

[6] D. Arcelli, V. Cortellessa, and C. Trubiani, "Antipattern-based model refactoring for software performance improvement," in Proceedings of the 8th international ACM SIGSOFT conference on Quality of Software Architectures. ACM, 2012, pp. 33–42.

[7] C. Trubiani and A. Koziolek, "Detection and solution of software performance antipatterns in palladio architectural models." in ICPE, 2011, pp. 19–30.

[8] J. Xu, "Rule-based automatic software performance diagnosis and improvement," Performance Evaluation, vol. 69, no. 11, 2012, pp. 525–550.

[9] D. Arcelli and V. Cortellessa, "Software model refactoring based on performance analysis: better working on software or performance side?" in Proceedings 10th International Workshop on Formal Engineering Approaches to Software Components and Architectures, Rome, Italy, March 23, 2013, ser. Electronic Proceedings in Theoretical Computer Science, B. Buhnova, L. Happe, and J. Kofroň, Eds., vol. 108. Open Publishing Association, 2013, pp. 33–47.

[10] Virgil. [Online]. Available: http://en.wikipedia.org/wiki/Virgil [retrieved: 08, 2014]

[11] C. Heger, "Systematic guidance in solving performance and scalability problems," in WCOP '13: Proceedings of the 18th international doctoral symposium on Components and Architecture. New York, NY, USA: ACM, 2013, pp. 7–12.

[12] O. M. Group. Business process model and notation (bpmn). [Online]. Available: http://www.omg.org/spec/BPMN/2.0 [retrieved: 08, 2014]

[13] C. Heger and R. Heinrich, "Deriving work plans for solving performance and scalability problems," in EPEW. Springer, 2014, pp. 104–118, (in press).

[14] T. Parsons, "Automatic detection of performance design and deployment antipatterns in component based enterprise systems," Ph.D. dissertation, University College Dublin, 2007.

[15] Jamopp. [Online]. Available: http://www.jamopp.org [retrieved: 08, 2014]

[16] S. Becker, H. Koziolek, and R. Reussner, "The palladio component model for model-driven performance prediction," Journal of Systems and Software, vol. 82, no. 1, 2009, pp. 3–22.

 ISBN: 978-1-61208-367-4

[17] S. Becker, M. Hauck, M. Trifu, K. Krogmann, and J. Kofron, "Reverse engineering component models for quality predictions," in Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on. IEEE, 2010, pp. 194–197.

[18] A. Van Hoorn, M. Rohr, and W. Hasselbring, "Generating probabilistic and intensity-varying workload for web-based software systems," in Performance Evaluation: Metrics, Models and Benchmarks. Springer, 2008, pp. 124–143.

[19] A. Wert, J. Happe, and L. Happe, "Supporting swift reaction: Automatically uncovering performance problems by systematic experiments," in Proc. of the 35th ACM/IEEE Int'l Conference on Software Engineering, ser. ICSE '13. New York, NY, USA: ACM, 2013, pp. 552–561.

[20] V. Cortellessa, A. Di Marco, and C. Trubiani, "An approach for modeling and detecting software performance antipatterns based on first-order logics," Software and Systems Modeling, 2012, pp. 1–42.

[21] M. L. Drago, "Quality driven model transformations for feedback provisioning," Ph.D. dissertation, Italy, 2012.

[22] A. Diaz-Pace, H. Kim, L. Bass, P. Bianco, and F. Bachmann, "Integrating quality-attribute reasoning frameworks in the arche design assistant," in Quality of Software Architectures. Models and Architectures. Springer, 2008, pp. 171–188.

[23] D. Arcelli, V. Cortellessa, and D. Di Ruscio, "Applying model differences to automate performance-driven refactoring of software models," in Computer Performance Engineering. Springer, 2013, pp. 312–324.

[24] M. Drago, C. Ghezzi, and R. Mirandola, "A quality driven extension to the qvt-relations transformation language," Computer Science - Research and Development, 2011, pp. 1–20.

[25] HP LoadRunner. [Online]. Available: http://www.hp.com/go/loadrunner [retrieved: 08, 2014]

[26] JMeter. [Online]. Available: https://jmeter.apache.org [retrieved: 08, 2014]

[27] Markov4JMeter. [Online]. Available: http://www.se.informatik.uni-kiel.de/en/research/projects/markov4jmeter/ [retrieved: 08, 2014]

[28] S. Lehnert, Q. Farooq, and M. Riebisch, "Rule-based impact analysis for heterogeneous software artifacts," in Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on. IEEE, 2013, pp. 209–218.

[29] ——, "A taxonomy of change types and its application in software evolution," in Engineering of Computer Based Systems (ECBS), 2012 IEEE 19th International Conference and Workshops on, April 2012, pp. 98–107.

[30] F. Moges Kasie, "Combining simple multiple attribute rating technique and analytical hierarchy process for designing multi-criteria performance measurement framework," Global Journal of Researches In Engineering, vol. 13, no. 1, 2013.

[31] T. L. Saaty, "The analytic hierarchy and analytic network processes for the measurement of intangible criteria and for decision-making," in Multiple criteria decision analysis: state of the art surveys. Springer, 2005, pp. 345–405.

[32] W. Edwards, "How to use multiattribute utility measurement for social decisionmaking," Systems, Man and Cybernetics, IEEE Transactions on, vol. 7, no. 5, 1977, pp. 326–340.

[33] A. Koziolek, H. Koziolek, and R. Reussner, "Peropteryx: automated application of tactics in multi-objective software architecture optimization," in Proceedings of the joint ACM SIGSOFT conference (QoSA+ISARCS'11). New York, NY, USA: ACM, 2011, pp. 33–42.

[34] Google Guava-Libraries. [Online]. Available: http://code.google.com/p/guava-libraries/ [retrieved: 08, 2014]

[35] V. Cortellessa, A. Martens, R. Reussner, and C. Trubiani, "A process to effectively identify "guilty" performance antipatterns," in Fundamental Approaches to Software Engineering. Springer, 2010, pp. 368–382.

[36] V. Cortellessa, A. Di Marco, and C. Trubiani, "Performance antipatterns as logical predicates," in Engineering of Complex Computer Systems (ICECCS), 2010 15th IEEE International Conference on. IEEE, 2010, pp. 146–156.

[37] A. Martens, H. Koziolek, S. Becker, and R. Reussner, "Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms," in Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering. New York, NY, USA: ACM, 2010, pp. 105–116.

[38] S. Malek, M. Mikic-Rakic, and N. Medvidovic, "An extensible framework for autonomic analysis and improvement of distributed deployment architectures," in Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems, ser. WOSS '04. New York, NY, USA: ACM, 2004, pp. 95–99.

[39] A. Sage, "Observation-driven configuration of complex software systems," 2010.

[40] P. Lengauer and H. Mössenböck, "The taming of the shrew: Increasing performance by automatic parameter tuning for java garbage collectors," in Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering, ser. ICPE '14. New York, NY, USA: ACM, 2014, pp. 111–122.

[41] T.-H. Chen et al., "Detecting performance anti-patterns for applications developed using object-relational mapping," in Proceedings of the 36th International Conference on Software Engineering, ICSE, 2014, pp. 1001–1012.

[42] B. W. Boehm, Software Engineering Economics, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1981.

 ISBN: 978-1-61208-367-4

A Domain-Specific Language for Modeling Performance Testing

Requirements Analysis and Design Decisions

Maicon Bernardino, Avelino F. Zorzo, Elder Rodrigues, Flávio M. de Oliveira
Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre, RS, Brazil
bernardino@acm.org, eldermr@gmail.com, {avelino.zorzo, flavio.oliveira}@pucrs.br

Rodrigo Saad
Dell Computers of Brazil Ltd.
Porto Alegre, RS, Brazil
rodrigo_saad@dell.com

***Abstract*—Performance is a fundamental quality of software systems. The focus of performance testing is to reveal bottlenecks or lack of scalability of a system or an environment. However, usually the software development cycle does not include this effort on the early development phases, which leads to a weak elicitation process of performance requirements. One way to mitigate that is to include performance requirements in the system models. This can be achieved by using Model-Based Testing (MBT) since it enables to aggregate testing information in the system model since the early stages of the software development cycle. This also allows to automate the generation of test artifacts, such as test cases or test scripts, and improves communication among different teams. In this paper, we present a set of requirements for developing a Domain-Specific Language (DSL) for modeling performance testing of Web applications. In addition, we present our design decisions in creating a solution that meets the specific needs of a partner company. We believe that these decisions help in building a body of knowledge that can be reused in different settings that share similar requirements.**

***Keywords*—performance testing; domain-specific language.**

I. Introduction and Motivation

Performance testing can be applied to improve quality of a Web-based service or application hosted on cloud computing or virtualization environments, since it supports the verification and validation of performance requirements [1]. Furthermore, it also supports evaluation of infrastructure's resource consumption while the application is under different workloads, *e.g.*, to accurately measure the resources required by an application that will respect the established Service Level Agreements (SLA). Despite the fact that performance testing is a well-known technique to validate performance requirements of an application or service, there is a lack of a modeling standard or/and language to support the specific needs of the performance testing domain.

Nevertheless, there are some notations, languages, and models that can be applied to represent a system behavior, *e.g.*, UML (Unified Modeling Language) [2], UCML (User Community Modeling Language) [3], CBMG (Customer Behavior Modeling Graph) [4], and WebML [5]. Some available modeling notations, *e.g.*, UML testing profiles, rely on the use of textual annotations on models, *i.e.*, stereotypes and tags, to support the modeling of performance aspects of an application. The use of notations, languages or models improve the performance testing activities, *e.g.*, reducing misinterpretation and providing a common document to stakeholders, system analysts and testers. Moreover, the use of a well-defined and concise notation, language or model, can support the use of Model-Based Testing (MBT) to generate inputs to the performance testing automation process, *e.g.*, test data, test scenarios and scripts can be automatically generated [6].

However, despite the benefits of using a UML profile to model specific needs of the performance testing domain, its use presents some limitations: (a) most of available UML design tools do not provide support to work with only those UML elements that are needed for a specialized language. Thus, the presence of unused and not required elements may result in an error-prone and complex activity; (b) UML diagrams are restricted to the semantics that is defined by Object Management Group (OMG) [2]. Therefore, in some cases the available UML elements and their semantics can restrict or even prevent the modeling of some performance characteristics of the Web domain.

It is important to highlight that UML is useful to analyze and design the architecture and the behavior of a system. Furthermore, it is a standard notation that does not imply in an implementation decision; besides, it is helpful for representing higher level concepts and the initial glossary domain. When compared to UML, Domain-Specific Languages (DSLs) are less general, and are based on an implementation strategy. That is, UML is used at an implementation independent level, whereas DSLs are used at an implementation dependent level. DSLs are restricted languages that can be used to directly model concepts in a specific problem domain. These languages can be textual, like most programming languages, or graphical. Furthermore, each DSL is a domain-specific code generator that maps domain-specific models into the required code.

In spite of the fact that performance testing is an active research field, researches investigating how to apply MBT approaches to automate the performance testing activities essentially started to be reported in the last decade and it is still on its early stages [6][7][8]. Furthermore, the lack of a standard to represent performance testing information is one of the major challenges to be explored from both, academic and industrial practitioners.

In this work, we discuss the requirements and design decision on the development of a modeling notation for performance testing. Thus, we propose a DSL that focus on meeting specific needs for modeling performance testing of Web applications. In this context, we also discuss the use of our DSL to support an MBT approach to generate performance testing artifacts to test these applications. Our contribution is twofold: (a) we identify a set of requirements, specific to our research context, that are not fully addressed by any known languages, model or notation. Thus, we elicit some practical scenarios that language providers and/or implementers may consider supporting; (b) we report our design decisions in supporting these requirements for an in-house solution. These decisions, in turn, may be reused or adapted to improve existing tools or devise new ones targeting similar needs.

This paper is organized as follows. Section II dis-

 ISBN: 978-1-61208-367-4

cusses background on performance testing, DSL and related work. Section III presents the context in which our work was developed. Section IV describes the domain analysis process. Section V enumerates the elicited requirements, which we address with specific design decisions, discussed in Section VI. Section VII briefly presents an example of use. Section VIII concludes the paper.

II. Background

A. Performance Testing

Software Performance Engineering (SPE) [9] describes and provides support to improve the performance through two distinct approaches: an early-cycle predictive model-based, and a late-cycle measurement-based. Performance testing is an essential measurement-based activity in the software development process, since it helps to identify bottlenecks that impact performance and scalability in a system. It is used to understand the behavior of a system under a certain workload.

According to Meier *et al.* [1], performance testing can be applied in different domains of applications, such as desktop, Web services, and Web applications. The process of designing and executing performance testing to a specific domain is composed by a set of well-defined activities. Therefore, to support the performance testing process, as well as its activities, a set of tools has been developed, *e.g.*, HP LoadRunner or Microsoft Visual Studio. Some of these tools support the generation of performance test scenarios and scripts through Capture and Replay technique or just manually coding scripts. Another technique that can be applied is MBT, which is useful to automate the performance testing process. A few academic and industrial performance testing tools based on models can be found in the literature, *e.g.*, SWAT [6], MBPeT [10] or PLeTs [11]. Despite the existence of some MBT tools, few of them use the same model, *i.e.*, a modeling standard for performance testing has not yet been set. Furthermore, there are some theoretical notations that do not allow test automation, *e.g.*, UCML [3].

B. Domain-Specific Language

DSLs, also called application-oriented, special purpose or specialized languages, are languages that provide constructs and notations tailored for a particular domain. DSLs are specific domain and problem-oriented computer languages [12]. DSLs are created to solve specific problems of a particular domain, *e.g*, in our case performance testing. However, to create a DSL, a domain analysis phase is required, which leads to a solid body of knowledge about the domain. During this phase, the domain's rules, features, and properties must be identified and documented. Currently, there are several tools, called Languages Workbenches (LWs), to support the creation and maintaining of a DSL, such as Eclipse Modeling Framework (EMF) [13], MetaEdit+ [14], among others. These tools are not restricted to analysis and code generation, LWs allow a DSL developer to create DSL editors with similar power to modern IDEs [12]. Thereby, a DSL can be classified in accordance with its creation techniques/design, that are the following: internal, external, and based on LWs.

Therefore, the use of DSLs presents some advantages, such as [15]: (a) better expressiveness in domain rules, allowing to express the solution at a high level of abstraction. Consequently, domain experts can understand, validate, modify or develop their own solutions; (b) improves the communication and collaboration among software project stakeholders; (c) supports artifacts and knowledge reuse. Inasmuch as DSLs can retain the domain knowledge, the adoption of DSL allows the reuse of the retained domain knowledge by a mass of users, including those that are not experts in the domain; (d) a DSL can provide a better Return Of Investment (ROI) than a traditional model. Despite that, the use of DSLs can present some disadvantages, such as [15]: (a) high cost to design and maintain a DSL, especially if the project presents a moderate or a high complexity. (b) high cost for training DSL users, *i.e.*, steep learning curve; (c) difficulty to define an adequate scope for a DSL; (d) a company could become dependent of an in-house language that is not used anywhere else; (e) in case of executable DSLs there are issues related to the performance loss when compared to source code written by a developer.

C. Related Work

The DSL community currently lacks evidence on the driving factors on the development of a DSL for the performance testing domain, its corresponding requirements and design decisions. There are few works describing DSLs requirements [16][17]. Whilst Kolovos [16] presents the core requirements and discuss the open issues with respect to the DSL requirements. Athanasiadis [17] describes the key requirements for a DSL to the environmental software domain, and discusses its benefits. However, these works present only an incipient discussion about the design decisions and, are not focused on the performance testing domain.

Conversely, there are some works [18][19] reporting design decisions for creating DSLs. Kasai *et al.* [18] proposed design guidelines that covers the following categories: language purpose, realization, content; concrete syntax; and abstract syntax. Frank [19] suggests guidelines to support the design decisions on a DSL development, *aka* DSML. Moreover, the author presents a useful discussion concerning the design decisions to DSLs. Although these studies are relevant, the design decisions were not taken to meet real requirements (*i.e* industrial requirements), but only based on the DSL creators knowledge and academic experience.

Some studies propose similar DSLs for the testing domain [20][21]. Bui [20] presents a DSL, DSLBench, for benchmark generation, while Spafford [21] presents a DSL, Aspen, for performance modeling. Different from our DSL proposal, focused in the measurement-based approach, Aspen supports the predictive-based approach. However, these works do not present any feedback from industrial cases where these DSLs are used, or discuss where they succeed or fail when applying on an MBT approach. Gatling [22] proposed an internal DSL based on industrial needs and tied to a testing tool. Unlike our proposal that provides a graphical DSL to represent the performance notation, Gatling provides only a textual DSL based on the Scala language. Moreover, our DSL is not tied or dependent of any performance testing tool or load generator. Wert *et al.* [23] present a novel automated approach for performance problem detection, in which they combined systematic search based on a decision tree with goal-oriented experimentation. However, this work is not based on MBT.

III. Context

This study is being conducted in cooperation with the Technology Development Lab (TDL) of a global IT company. This

 ISBN: 978-1-61208-367-4

cooperation aims to develop new strategies and approaches for software testing. The software process adopted by TDL is briefly described as follows. First the development teams implement the unity tests, which performs the preliminary tests in a dedicated development environment. After a stable version that meets its requirements is available, it is published in the test environment for the test team to perform functional and integration tests. If failures are found, the development team removes faults from the application and the process is restarted. If no failure is found, the software version becomes available for the performance testing team to perform load and stress testing in a new environment (called performance environment). It is important to mention that for some applications, sometimes the complexity or size of the production environment is so high that testing is executed only in a partial production environment and proportional estimations are used [10]. Finally, when the application meets the defined quality requirements, *e.g.*, response time, that version is deployed to the production.

Empirical evidences of our previous work [24], developed in collaboration with the TDL, indicate that a performance MBT approach is valuable to mitigate the effort to generate performance scripts. Our findings also indicate that performance testing teams, usually, choose notations and modeling tools by their own convenience. Thus, many models can be found across the company performance testing teams, which leads to a segmented knowledge about applications requirements. Hence, the use of different models can present other issues, such as inhibit communications among testing teams and increases the possibility of requirements misinterpretation. To attenuate these issues, it is necessary that a single notation is used across the entire performance testing division. Since there is not a standard modeling language to model the specific needs of the performance testing domain, we focused our effort on investigating and proposing a DSL to this domain.

IV. Domain Analysis

Before we start to describe our DSL, it is important to mention some of the steps that were taken prior to the definition of the requirements and design decisions. These steps were taken in collaboration with researchers from our group and test engineers from TDL.

The first step is related to the expertise that was acquired during the development of a Software Product Line (SPL) to generate MBT tools called PLeTs [11]. This SPL was split in two main parts: one to analyse models and generate abstract test cases from those models, and; another that would take the abstract test cases and derive actual test scripts to be executed by performance testing tools. Actually, our SPL is divided in four main features: parser, test case generation, script generation, and execution. Several models were studied during the first phase of the development of our SPL, *e.g.*, UCML, UML Profiles, Finite State Machines (FSM). Likewise, several performance testing environments and tools were studied, such as HP LoadRunner, Microsoft Visual Studio, among others.

The second step is related to the use of some of the above models and tools to actual applications, such as TPC-W and Moodle. Furthermore, we also used some of the products generated by our SPL to test those applications. Some of other real applications from our partner were also tested in the context of our collaboration. Those real applications were tested in very complex environments, which gave us a very thorough understanding of the needs a testing team has.

Besides that, we also based our DSL on well-known concepts from SWEBOK, IEEE Std. 610.12-1999, IEEE Std. 829-2008, and other literature, such as Meier *et al.* [1]. These references were chosen to mitigate the bias, provide a theoretical basis and ensure the coherency among concepts, features, and properties of the performance domain. The above steps provided us with a small Performance Testing Body Of Knowledge (PTBOK) that is used to define the performance testing requirements and design decisions for our DSL. Furthermore, prior to create any DSL to support modeling performance testing in the target DSL, the TDL first considered the use of off-the-shelf solutions, provided that specific requirements were met.

V. Language Requirements

This section enumerates the requirements we collected from our expertise and also from the software engineers from TDL. These requirements are related to features and concepts from performance testing domain. Moreover, we discuss some mechanisms for implementing the proposed DSL.

RQ1) *The DSL must allow to represent the performance testing features.* One of the main functions of the performance testing is to reveal bottlenecks of a system. Therefore, the applications should be measured and controlled in small parts that can be defined as transactions. This allows to measure the performance quality for each activity of a system. For instance, to define the response time SLA based on these transactions.

RQ2) *The technique for developing our DSL must be based on LW.* Since we do not want to develop new tools, *i.e.*, editor or compiler, as in an external DSL; neither we intend to embed our DSL in a GPL, we will base our DSL on a LW. This will allow us to focus on the analysis domain and development of the new DSL rather than spend effort on implementing new tools or having to choose a GPL language that might not be appropriate for the DSL that we want.

RQ3) *The DSL must support a graphical representation of the performance testing features.* This requirement does not concern the language itself, but the LW that will support its development. Thereunto, we desire that the LW supports a graphical-based editor for creating DSLs. Moreover, the LW should allow to implement the domain concepts, their translation rules, designing symbols and elements of the language, and also to generate different code for different tools.

RQ4) *The DSL must support a textual representation.* The proposed DSL should also include a custom language that is close to a natural language. This will facilitate its adoption by test engineers that are used to use textual representation. The language should have features and keywords that remember the performance testing domain.

RQ5) *The DSL must include features that illustrate performance counters.* In performance testing there are many performance counters, *e.g.*, response time or network throughput, that provide means to analyze both application quality level and host infrastructure.

RQ6) *The DSL must allow to model the behavior of different user profiles.* This requirement is a specific function of the performance domain, which should allow that the behavior of different user profiles, such as a buyer or a new clients, is modeled according to the System Under Test (SUT). In our context we will focus on Web applications.

 ISBN: 978-1-61208-367-4

RQ7) *Traceability links between graphical and textual representations should require minimal human intervention/effort.* Traceability is an important feature in software solutions, mainly when involve model transformation, *e.g.*, translation from a graphical to a textual representation. The proposed DSL should automate the mapping process of graphical elements of the model to their respective textual counterparts.

RQ8) *The DSL must be able to export models to formats of specific technologies.* This requirement should ensure that models written in our proposed DSL can be exported to the format of the input of specific testing tools, *e.g.*, HP LoadRunner, MS Visual Studio or Apache JMeter.

RQ9) *The DSL must generate model information in a eXtensible Markup Language (XML) file.* This requirement aims to ensure that we can export our DSL to any other technology in the future. That is, we export all information from the system model into a XML file, so anyone that wants to use our solution can import the XML into their technology.

RQ10) *The DSL must represent different performance test elements in test scripts.* The modeled diagram using the proposed DSL must represent multiples elements of test scripts, such as conditional or repetition control flows, among others.

RQ11) *The DSL must allow the modeling of multiple performance test scenarios.* Performance testing is responsible to carry out testing of part of or the whole system under normal and/or stress workload. The DSL, therefore, should be able to generate multiples performance test scenarios, *i.e.*, under normal and stress workload conditions.

Currently, to the best of our knowledge, no existing language or model (commercial or not) meets all of the presented requirements. Therefore, given the automation needs of performance testing, we propose a DSL for modeling performance testing of Web applications.

VI. Design Decisions

In this section, we describe our design decisions for creating a DSL that supports the requirements discussed in Section V. For each design decision, we mention the associated requirements that are being dealt with.

DD1) *To use a LW that supports graphical DSLs* (RQ2, RQ3). To attend these requirements we performed a literature review on existing LWs, including academic, commercial or open-source. The work of Erdweg *et al.* [25] presents the state-of-the-art in LWs and defines some criteria (the authors call them features) that help someone to decide which tool should be adopted. Given the requirements of our proposed DSL, we chose the MetaEdit+ from MetaCase [14], because it supports most of the features evaluated by work.

DD2) *The features of the performance testing domain will be used in an incremental way* (RQ1, RQ5). Developing a DSL requires a series of phases, such as analysis, design, implementation, and use [26]. Usually researchers focus their attention to the implementation phase, but only a few of them focus on the analysis of the domain and design of the DSL. Nevertheless, there are some methodologies for domain analysis, which helps to unravel the knowledge about the problem domain analyzed. Among them we can highlight Domain Specific Software Architectures (DSSA), Feature-Oriented Domain Analysis (FODA), and Organization Domain Modeling (ODM). Some works present an approach based on the formalization of domain analysis through ontologies [27][28]. Thus, in order to determine the features that represent the performance testing domain, we adopted a strategy to identify and analyze the domain using an ontology [29]. This ontology provides the basis for determining the concepts, relationships, and constraints that represent the performance testing domain. Besides the ontology, we have used the PTBOK (Section IV).

DD3) *To provide a graphical language capable of representing the behavior of user profiles for different performance test scenarios* (RQ6, RQ11). To attend these requirements we analysed different models and graphical representations that support performance testing. Among the approaches and techniques, the most relevant for our work were UML profiles. Besides that, it is also important to mention a theoretical language proposed by Scott Barber for modeling users behavior, called UCML. Based on these different approaches and techniques, the graphical language will have visual elements capable of representing the behavior of different user profiles. Besides the flow of activities that the user performs in the SUT, the graphical language will have visual elements to represent the performance test scenarios settings, including information about the performance testing domain, such as number of Virtual Users (VU), test duration, metrics to be evaluated (response time, memory available, processor time, among others). It is also possible to include the randomization and execution probabilities for each interaction that a VU executes during performance testing. Another very important feature is that the DSL can represent abstract data that will be instantiated in activity of the performance testing process, for example, during the generation of the performance test scripts.

DD4) *To create a textual representation in a semi-natural language* (RQ4). Behavior-Driven Development (BDD) [30] is an agile software development process, in which acceptance testing, mainly functional testing, is essential to advance to next phase of a software project, since it facilitates the understanding among testing and development teams and stakeholders. Usually, tests are described in natural language in order to ensure this common understanding regarding the system requirements for all project members. Even though it is common to find languages that use natural language to describe functional testing, *e.g.*, Gherkin [30], to the best of our knowledge none of them includes performance testing features. Therefore, we intend to extend this language, to include the performance testing features described in Section IV. Gherkin is interpreted by a command line tool called Cucumber, which automates the acceptance testing execution.

DD5) *To provide automated traceability between the graphical and textual representations* (RQ7, RQ10). Traceability is an important feature that should be mapped in the implementation of a DSL. Thus, it is required that the LW allows the creation of translation rules among models. In this case, the mapping among the graphical elements with their respective assets of the textual representation must be provided. It is important that this mapping is not an one-to-one mapping. Some graphical elements can be mapped to several instances of the textual elements. For example, a graphical decision point can be mapped to several textual scripts, one for each branch present in the graphical representation. In order to solve this mapping, algorithms such as the Chinese Postman Problem can be used.

DD6) *To support the integration of the DSL with other technologies* (RQ8, RQ9). It should be able to export the models (test scenarios, abstract test cases, etc.) described in the DSL to other formats, such as XML or HP LoadRunner

 ISBN: 978-1-61208-367-4

and MS Visual Studio input formats. The ability to export data in XML format will allow future users of the language to use it with other technologies or IDEs.

VII. DSL for Modeling Performance Testing

This section presents the DSL we developed to meet the requirements described in Section V and based on the design decision from Section VI. Our DSL is composed of three parts: monitoring, scenario, and scripting.

Monitoring: The performance monitoring part is responsible for determining all servers used in the performance testing environment. For each server (*i.e.*, application, databases, or even the load generator), information on the actual testing environment has to be included, *e.g.*, IP address or host name. It is worth mentioning that even the load generator has to be described in our DSL, since we can also monitor the performance of the load generator. Sometimes, the load generator has to be split in several servers if we really want to stress the application or database server. For each host, it is possible to indicate the performance counters that will be monitored. This monitoring part requires that at least two servers have to be described: one that hosts the application (SUT) and another to generate the workload and monitor the performance counters of the SUT.

Scenario: The performance scenario part allows to set user and workload profiles. Each user profile is associated to test scripts. If a user profile is associated with more than one test script, a probability is attributed between the user profile and each test script, *i.e.*, it describes the probability that that test script is executed. In addition to setting user profiles, in this part, it also is important to set one or more workload profiles. Each workload profile of is composed of several elements, defined as follows: (a) *virtual users*: number of VU who will make requests to the SUT; (b) *ramp up time*: time it takes for each set of ramp up users to access the SUT; (c) *ramp up users*: number of VU who will access the SUT during each ramp up time interval; (d) *Test duration*: refers to the total time of performance test execution for a given workload; (e) *ramp down users*: defines the number of VU who will left the SUT on each ramp down time; (f) *ramp down time*: defines the time it takes for a given ramp down user stop the testing.

Scripting: The performance script part represents each of the test scripts from the user profiles in the scenarios part. This part is responsible for determining the behavior of the interaction between VU and SUT. Each test script includes activities, such as transaction control or think time between activities. The same way as there is a probability for executing a test script, which is defined in the scenarios part, each test script can also contain branches that will have a user distribution associated to each path to be executed, *i.e.*, the number of users that will follow each path. During the description of each test script it is also possible to define a decision table associated to each activity. This decision table [12] represents the decisions that is taken by a user based on the response that an application provides. Actually, the decision table will be used to guarantee that the user distribution rate is achieved.

A. Example of Use: TPC-W

This section presents a small sample of the graphical representation described in previous sections. We instantiate the modeling performance testing process through the proposed DSL for the TPC-W e-commerce Web application. The goal

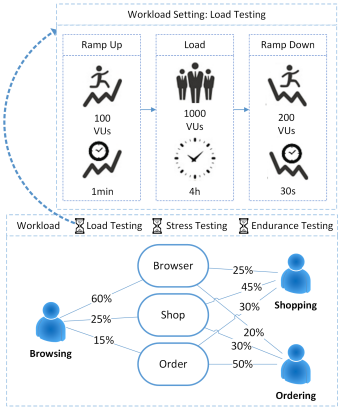


Figure 1. Graphical representation of a performance test scenario model.

is to give some understanding of the requirements and design decisions of a language to model user behavior, as well as the performance test scenarios. The graphical representation contains elements for virtual users, test duration, ramp up time, ramp up users, ramp down time, think time, among others.

Figure 1 gives an example of a performance test scenario model that represents the SUT functionalities divided into three scripts: `Browser`, `Shop` e `Order`. Each script has a percentage of the VU that will execute such script. For each script, a VU has a probability of buying a product from the site. This probability is bigger for script `Order` and smaller for script `Browser`. Due to space limitation this is not shown in this paper. The model also shows the interaction behavior of three different user profiles: `Browsing`, `Shopping` e `Ordering`. Basically, the user profiles differ from one another on the probability that they will order, shop or browse.

A snippet of the `Browser` script is presented in Figure 2. The model is composed of six activities, five of them with transaction control, shown by dashed border figures. The model also contains a `Think Time` of 15 seconds (Clock element) and three `Data Table`, *e.g.*, `Transaction Data`. In some cases, there is the necessity to store the results of processing of an activity into global variables or parameters, so that this data can be used in other activities, for example to decide a path in a decision point (see `Category choice`). The model also shows a transaction composed by a set of activities, see the `Search Products` transaction, which is composed of `Search request` and `Search result`.

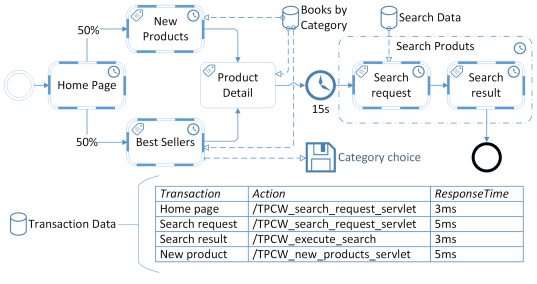


Figure 2. Performance test scripting model of the `Browser` script.

VIII. Lessons Learned and Final Remarks

Basically, the main lessons we have learned from the requirements and design decisions of our DSL are: (a) there

 ISBN: 978-1-61208-367-4

are several techniques to achieve the best possible results from requirements elicitation process, such as interviews, ethnography, domain analysis, among others. We have used domain analysis to understand and capture our domain knowledge, and to identify of reusable concepts and components. Thus, we learned that eliciting requirements based on domain analysis, having as output the PTBOK, was effective to identify what we needed for our DSL; (b) domain analysis based on ontologies is a good alternative to transform the concepts and relationships from the ontology into entities and functionalities of the DSL. There are several methods and techniques for describing this approach, for instance [27][28]; (c) one of the disadvantages of using DSLs is the high cost of training users who will use the DSL, *i.e.*, steep learning curve [15]. However, based on our previously experience using several load generator tools in an industrial setting, this disadvantage can be handled pragmatically, since the cost for a new staff to learn several load generators technologies is higher than compared to our DSL. Nonetheless, this drawback must be proved with empirical evidences; (d) Global Software Development refers to software development geographically or globally distributed, which aims to streamline the process of product development. In such scenario it is common that infrastructure and performance teams are located in different countries. For this reason, it is important to adopt a standard language for creating scripts and models for performance testing, hence we chose the English as default for the textual representation of our DSL, implicitly we avoid a cacophonous language [12]; (e) we adopt an incremental development methodology for creating our proposed DSL. This methodology allows us to improve the DSL on each interaction, which is composed by the following steps: analysis, development, and utilization [15].

This paper presented a set of requirements elicited in the context of an industrial partner. Through a pilot study and based on our PTBOK, we collected specific needs for performance modeling adoption to create a DSL for modeling performance testing, and argue that existing models and languages do not meet the specificity of the requirements at hand. We then presented our design decisions for creating a DSL. We claim that the reported requirements and design decisions as the two contributions of this work, since currently few studies bring such discussion. Our work adds to that in the sense that the elicited requirements evidence practical scenarios that other load generators may consider supporting; the design decisions, in turn, may be reused or adapted to improve existing DSLs, models or languages, or even new ones, targeting similar requirements.

IX. Acknowledgment

Study developed in the context of PDTI 001/2014, financed by Dell Computers with resources of Law 8.248/91.

References

[1] J. Meier, C. Farre, P. Bansode, S. Barber, and D. Rea, Performance Testing Guidance for Web Applications: Patterns & Practices. Microsoft Press, 2007.

[2] OMG, "Object Management Group," 2014, URL: http://www.omg.org [retrieved: 08, 2014].

[3] S. Barber, "User Community Modeling Language (UCML) for performance test workloads," Sep. 2003, URL: http://www.perftestplus.com/articles/ucml.pdf [retrieved: 08, 2014].

[4] D. Menascé, V. Almeida, R. Fonseca, and M. Mendes, "A Methodology for Workload Characterization of E-commerce Sites," in 1^{st} ACM Conference on Electronic Commerce. ACM, 1999, pp. 119–128.

[5] N. Moreno, P. Fraternali, and A. Vallecillo, "WebML modelling in UML," IET Software, vol. 1, no. 3, pp. 67–80, June 2007.

[6] M. Shams, D. Krishnamurthy, and B. Far, "A Model-based Approach for Testing the Performance of Web Applications," in 3^{rd} International Workshop on Software Quality Assurance, 2006, pp. 54–61.

[7] D. Krishnamurthy, M. Shams, and B. H. Far, "A Model-Based Performance Testing Toolset for Web Applications," Engineering Letters, vol. 18, no. 2, pp. 92–106, 2010.

[8] M. B. da Silveira *et al.*, "Generation of Scripts for Performance Testing Based on UML Models," in 23^{rd} International Conference on Software Engineering and Knowledge Engineering, Jul. 2011, pp. 258–263.

[9] M. Woodside, G. Franks, and D. C. Petriu, "The Future of Software Performance Engineering," in Future of Software Engineering, 2007, pp. 171–187.

[10] F. Abbors, T. Ahmad, D. Truscan, and I. Porres, "MBPeT - A Model-Based Performance Testing Tool," in 4^{th} International Conference on Advances in System Testing and Validation Lifecycle, 2012, pp. 1–8.

[11] E. M. Rodrigues, L. D. Viccari, A. F. Zorzo, and I. M. Gimenes, "PLeTs Tool - Test Automation using Software Product Lines and Model Based Testing," in 22^{th} International Conference on Software Engineering and Knowledge Engineering, Jul. 2010, pp. 483–488.

[12] M. Fowler, Domain Specific Languages, 1st ed. Addison-Wesley, 2010.

[13] EMF, "Eclipse Modeling Framework," 2014, URL: http://www.eclipse.org/modeling/emf/ [retrieved: 08, 2014].

[14] S. Kelly, K. Lyytinen, and M. Rossi, "MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment," in 8^{th} International Conference on Advances Information System Engineering. Springer, 1996, pp. 1–21.

[15] D. Ghosh, "DSL for the Uninitiated," Queue, vol. 9, no. 6, pp. 10–21, 2011.

[16] D. Kolovos, R. Paige, T. Kelly, and F. Polack, "Requirements for Domain-Specific Languages," in 1^{st} Domain-Specific Program Development, Jul. 2006, pp. 1–4.

[17] I. N. Athanasiadis and F. Villa, "A Roadmap to Domain Specific Programming Languages for Environmental Modeling: Key Requirements and Concepts," in ACM Workshop on Domain-Specific Modeling, 2013, pp. 27–32.

[18] G. Karsai *et al.*, "Design Guidelines for Domain Specific Languages," in 9^{th} OOPSLA Workshop on Domain-Specific Modeling, 2009, pp. 7–13.

[19] U. Frank, "Domain-Specific Modeling Languages: Requirements Analysis and Design Guidelines," in Domain Engineering, 2013, pp. 133–157.

[20] N. Bui, L. Zhu, I. Gorton, and Y. Liu, "Benchmark Generation Using Domain Specific Modeling," in 18^{th} Australian Software Engineering Conference, Apr. 2007, pp. 169–180.

[21] K. L. Spafford and J. S. Vetter, "Aspen: A Domain Specific Language for Performance Modeling," in International Conference on High Performance Computing, Networking, Storage and Analysis, 2012, pp. 1–11.

[22] Gatling, "Gatling Stress Tool," 2014, URL: http://gatling-tool.org [retrieved: 08, 2014].

[23] A. Wert, J. Happe, and L. Happe, "Supporting Swift Reaction: Automatically uncovering performance problems by systematic experiments," in 35th International Conference on Software Engineering, May 2013, pp. 552–561.

[24] E. M. Rodrigues *et al.*, "Evaluating Capture and Replay and Model-based Performance Testing Tools: An Empirical Comparison," *"forthcoming"* in International Symposium on Empirical Software Engineering and Measurement, 1–8 2014.

[25] S. Erdweg *et al.*, "The State of the Art in Language Workbenches," in Software Language Engineering, 2013, vol. 8225, pp. 197–217.

[26] M. Mernik, J. Heering, and A. M. Sloane, "When and How to Develop Domain-Specific Languages," ACM Computing Surveys, vol. 37, no. 4, pp. 316–344, 2005.

[27] R. Tairas, M. Mernik, and J. Gray, "Models in Software Engineering," M. R. Chaudron, Ed. Springer, 2009, ch. Using Ontologies in the Domain Analysis of Domain-Specific Languages, pp. 332–342.

[28] T. Walter, F. S. Parreiras, and S. Staab, "OntoDSL: An Ontology-Based Framework for Domain-Specific Languages," in 12^{th} International Conference on Model Driven Engineering Languages and Systems, 2009, pp. 408–422.

[29] A. Freitas and R. Vieira, "An Ontology for Guiding Performance Testing,," *"forthcoming"* in International Conferences on Web Intelligence and Intelligent Agent Technology, 2014, pp. 1–8.

[30] M. Wynne and A. Hellesoy, The Cucumber Book: Behaviour-Driven Development for Testers and Developers. The Pragmatic Bookshelf, Jan. 2012.

 ISBN: 978-1-61208-367-4

Inverted Run-Time Behavior of Classic Data Structures on Modern Microprocessors: Technical Background and Performance Guidelines

Michael Bogner, Andreas Hofer, Maria Hronek, Franz Wiesinger

University of Applied Sciences Upper Austria
Department of Embedded Systems Engineering
Hagenberg, Austria
Email: {michael.bogner, andreas.hofer, maria.hronek, franz.wiesinger}@fh-hagenberg.at

***Abstract*—Classic data structures, such as vectors and lists are used for storage and organization of data. Certain basic operations have a specified run-time behavior, which is essentially influenced by the choice of the data structure. However, further advances in the development of modern microprocessors have achieved sophisticated optimizations in hardware. These optimizations affect the run-time behavior of certain operations, which further affects the choice of data structures. This paper presents the results of our research activities focused on the impacts of these changed conditions. We selected various algorithms and operations frequently used in todays software development. Remarkable differences and modified characteristics will be discussed. The performances of both selected data structures, namely vector and list, have been determined empirically using the programming language C++. The results are interpreted and discussed in terms of run-time complexity and modern processor development.**

Keywords–performance patterns; sequence container; run-time complexity; modern microprocessors;

I. INTRODUCTION

The performance of microprocessors increased greatly in recent years. This was possible mainly because of sophisticated optimizations in processor architecture, achieved by modern processor development. One substantial reason for the high processing speed of modern Central Processing Units (CPUs) is the hierarchy of various storage levels of cache memory on the processor die. But, in our consideration, also the main memory is important. It is critical to organize data as efficiently as possible in the main memory in order to get maximum performance. Different container data structures allow us to select the appropriate organization. However, not all container types benefit in the same way of modern processor architecture, which leads to different performance gains. Under these changed conditions, current approaches need to be reconsidered.

For the comparison in this paper, we have selected data structures from the Standard Template Library (STL) of the widely used C++ programming language. As performance is a key aspect, we decided to avoid languages which have managed runtime features or rely on virtual machine support, or use Just-In-Time compilation (JIT) and garbage collection. C++ compiles to native code, is platform independent and comes with efficient container implementations of the STL, which substantially reduces interfering side effects.

The selected containers are std::vector and std::list. Both are basic sequence containers, but as generally known, rely on completely different implementations. The std::vector uses a strict byte-sequence and therefore guarantees a contiguous storage space in memory [1]. In contrast, the std::list is a doubly linked list which represents the simplest form of a graph-based data structure, except for the rarely used single linked list. This is a significant difference for the microprocessor, and this paper analyses right this aspect. Advanced data structures like sets and trees are variants of these basic implementation types. Usually, they use hash functions or other optimizations to gain algorithmic benefits. But these algorithmic optimizations are not directly related to the performance of the microprocessor.

The theory teaches that each data structure has its advantages depending on the scenario. Operations such as inserting, deleting, or accessing elements have a proven run-time behavior, which is described by the asymptotic run-time complexity or the big $\mathcal{O}$-notation [2]. This notation makes it possible to specify the run-time as a function of the problem size. Or in other words algorithms can be classified by how they respond to changes in the container size. In software development, this classification essentially influences the choice of data structure.

For the reasons mentioned above, this paper investigates whether classical selection criteria for list and vector, described by the $\mathcal{O}$-notation are still valid or not. In order to achieve the objective, prototype implementations on both data structures were evaluated. The performances were determined empirically on a common workstation; for details, see Section III-B. To measure the run-time and the run-time behavior, various algorithms and operations were selected, which are frequently used in today software development. The results were interpreted and discussed in terms of run-time complexity and modern processor development. Finally, the main rules that can be held responsible for the results are filtered out, to get general performance patterns. These performance patterns represent some guidelines for todays software development on modern microprocessors.

Already Niklaus Wirth noted the importance of data structures in order to create effective applications [3], as data organization is highly relevant. There are quite a lot text books and papers introducing elementary and advanced data structures and corresponding algorithms like [4], including its $\mathcal{O}$-notation. Professional usage of the C++ standard template library containers and their performance guarantees are covered by Musser et al. [5]. To analyse already existing applications, Liu and Rus [6] present a tool for detecting poor data structure selection in C++ programs, which gives a context sensitive performance advice. An automated tool to

 ISBN: 978-1-61208-367-4

generate cost models of given data structures is introduced by Jung [7]. After a training phase to understand the effect of architectural behavior, the statistical data are then fed into a machine learning model which tries to determine the optimal data structure. Its vision is to build it into data structure libraries so that the compiler can automatically select the best implementation.

The paper has the following structure: The first Section presents a short introduction. Section II describes the development of modern processor architectures and why that prefers specific data structures. Section III presents the methodological approach of the test procedure. Also the selected performance tests are presented. The most distinctive test results are presented and analyzed in Section IV. Section V shows the main rules that can be held responsible for the results, whereas the final thoughts are presented in Section VI.

II. MODERN PROCESSOR ARCHITECTURES

This section gives important background information that can be made responsible for the special effects discussed in this paper. These include a short introduction of the properties of modern processor architectures and their effect on classic data structures. Especially, the memory architecture and how modern CPUs optimize memory accesses are points which are addressed, because almost exclusively array data structures benefit from certain optimizations of modern microprocessors, as we see below. Particularly, in view that the memory access is a growing bottleneck, such things can be critical and lead to significant shifts in the performance analysis.

According to Moore's Law, the integration density of transistors on integrated circuits doubling every two years [8] [9], which made more complex and fast CPUs possible. Thanks to a better understanding of the architecture, the Instruction Level Parallelism (ILP) and increasing clock speeds allows to raise the processor performance considerably long time. Described by ILP, the executed Instructions Per Clock cycle (IPC) of the processor were increased. This was achieved by techniques such as pipelining, super scalarity, out-of-order-execution, branch-prediction or speculative execution [10].

One of the biggest challenges during this development was to design memory systems that can provide the processor fast enough with data. Because the increase in speed of modern processors is not accompanied by a corresponding acceleration of the memory systems. This means that memory access is quite slow in relation to processor performance. Figure 1 shows the development of memory and processor performance since 1980 - note the logarithmic scaling in this diagram to be able to display the large gap in this development. The memory line starts with 64 KiB DRAM 1980 and reached an annual latency increase of factor 1.07. The speed increase at the processors reached factor 1.25 until 1986, 1.52 until 2004 and 1.2 after 2004 [11].

Not only the latency, but also the bandwidth to the memory system is important for the CPU. These two points stay in strong conflict to each other. Therefore, a technique for increasing the memory bandwidth often results in an increase of latency, and vice versa. The higher the speed of processors grows, the harder it is to realize a memory, which can provide

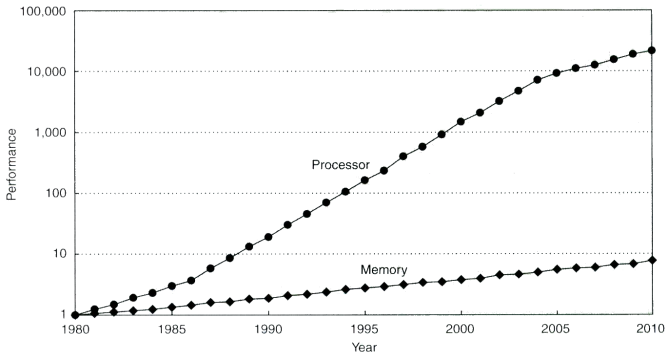


Figure 1. Performance of processors and latency of the memory plotted over time [11].

data fast enough in a few clock cycles. Therefore, the memory system was an increasing bottleneck [12].

To minimize this problem at least, modern processors have various cache levels on die. The purpose of the cache is to take the last used memory words, whereby the new access to them is greatly accelerated. If a sufficiently large amount of the required data is present in the cache, the effective memory latency shrinks enormously. This advantage is tried to maximize with several cache levels [12].

To reach their target, caches feature a variety of optimization; many of them use the memory address of the corresponding data (address locality): The so-called spatial locality refers to the observation that memory locations, that are numerically similar to locations which were accessed recently, will be accessed in the near future with increased probability. This property is exploited by caches reading more data than requested, in the assumption to predicting future accesses. Such optimizations are called prefetching, as data will be already prefetched from the main memory [12]. In terms of the performance tests in this paper, it should be noted that almost exclusively array data structures benefit from these optimizations, because of their contiguous memory order. For lists with a scattered memory order, such optimizations are nearly useless.

III. METHODOLOGICAL APPROACH

This section shows the methodological approach performing the test. The selected tests are described and their purpose is explained. Also, the test system is presented with the hardware and software base. Finally, criteria of the test procedure are determined.

A. Selected performance tests

In principle, we selected various algorithms and operations frequently used in todays software development. We tried to figure out the respective advantages and disadvantages of the two data structures on an modern microprocessor and want to illustrate if classical selection criteria for list and vector are still valid. In Table I, the selected performance tests are presented. The first tests will cover basic operations, such as inserting or deleting elements. Subsequent test cases also check moving, comparing, swapping or sorting items in the containers - in

 ISBN: 978-1-61208-367-4

various combinations and executions. Not all test cases modify the size of the data structures; some only change the order of the data within the containers or are read-only tests. Details are noted in Table I.

TABLE I. SELECTED PERFORMANCE TESTS: EXECUTED TEST CASES AND THEIR DESCRIPTION AND PURPOSE

Test case	Description
Filling data structures	This is one of the most common operations in general. The elements in each of the newly declared containers are inserted at the back. In addition, also a vector is tested which gets the final size communicated via vector::reserve.
Clearing data structures	It is tested how quickly a filled container can be cleared completely. Since vector::clear does not change the capacity, for a fair comparison a new container is created using vector::swap.
Insert front	Also the insertion of elements on other positions are meaningful test scenarios. In this test case, additional elements will be inserted at the front of already filled containers.
Insert middle	It is measured how long it takes to insert additional elements in the middle of already filled data structures. The respective advantages and disadvantages of the two test candidates can be very well shown in this test case.
Insert sorted	In contrast to insert middle, in this test case the insertion point must be found first, so the elements in the data structures have to be accessed. The test starts with empty containers.
Reversing data structures	This test case reverses the order of the values in the containers. The first value becomes the last one, the second value the last but one etc. Therefore, the last value becomes the first value of the reversed container. It is shown how efficient elements in the containers can be swapped.
Is sorted	It is checked whether sorted containers are actually sorted. It just depends on how fast the data structures can be run through and accessed, making this text case clearly different from the previous.
Calculation of the arithmetic mean	The arithmetic mean is calculated over the containers. Similar to the previous point it must be iterated through the data structures. Not merely elements must be compared with each other, but also arithmetic operations occur.
Delete all occurrences of a number	In this test case, the deletion of specific numbers is measured. This process is repeated with all numerical values until the data structures are completely empty. Also different implementation variants are tested using the vector, to show the respective advantages and disadvantages.
Stable sort	A frequently occurring operation is also the sorting of data structures, which is represented by this test case. As the sorting algorithm of the std::list is stable, the vector uses also a stable sorting algorithm for a fair comparison.
Delete all duplicates	Every container is filled ten times in a row with the same number. It will be measured how long it takes to delete all duplicates. With one run though the data structures, the size is reduced to a tenth of its original value, what this test case clearly differ from the others.
Double each element	Each element in the containers is placed next to the current position again. Thereby the container size is doubling. The advantages and disadvantages of the vector compared to the list can be illustrated nicely in this test case.

B. Test system

The performance tests took place on the following system:

- Intel® Core™ Processor i5-3570K, 4 x 3.40GHz
- Corsair® Memory 16 GiB DDR3-1333 CL9
- Intel® Media Series DH77EB Mainboard
- Samsung® SSD 830 Series 256GB, SATA 6Gb/s
- Microsoft® Windows™ 7 Home Premium 64-bit with Service Pack 1 (March 2013)

The chosen test system represents a common workstation, no high-end device or special hardware. This should demonstrate the general validity of the test results on widespread available systems. The Intel® Core™ family has not changed significantly in those points relevant for the test, eg. from used Ivy Bridge model to the current Haswell architecture. The respective tests were implemented using Visual Studio 2010 Ultimate with Service Pack 1 and was compiled with compiler version 16.00.40219.01. Furthermore, the x64 version was used as a release build. The default settings of Visual Studio were used, with optimization level "O2 maximize speed".

C. Criteria of the test procedure

In order to ensure meaningful results, the following criteria have been defined:

- It was ensured that there is no main memory overflow. Because outsourcing of data on the hard drive would lead to significant performance degradation. The containers are small enough to find place in the main memory in any case.
- Dynamic frequency scaling (Enhanced Intel® SpeedStep® Technology (EIST) and Intel® Turbo Boost Technology) has been disabled. Therefore, the processor is running with the base clock of 3.4 GHz during the tests.
- Every test result has been repeated several times and the arithmetic mean has been extracted from the times of the measurements. This avoid measurement errors and reduce the possible impact of background processes from the operating system. Details will be shown in Section IV.
- To evaluate the run time behavior, five different container sizes were selected for each test case.
- The selected value type of each data structure is integer. Therefore, each element represents a four-byte signed value.

IV. PRESENTATION AND ANALYSIS OF THE RESULTS

Three test cases out of the twelve shown in section III-A are now presented and analyzed in detail. These most distinctive test cases, which provide particularly remarkable results are "Filling data structures", "Insert sorted" and "Stable sort". The results were interpreted and discussed in terms of run-time complexity and modern processor development.

A. Filling data structures

In this test case, the elements in each of the newly declared containers are inserted at the back. In addition, also a vector is tested which gets the final size communicated via vector::reserve, avoiding typical resize operations. The respective container sizes and the results are shown in Figure 2. In order to ensure sufficient runtime, each test ran 100 times and run times were summed up.

The vector dominates this comparison against the list. Beyond that the vector that has reserved all required memory before the measurement is nearly three times as fast as the normal vector. Both containers allow the insertion at the end with the run-time complexity of $\mathcal{O}(1)$, but there are significant differences in detail: the vector allocates memory always for multiple elements, to avoid of requesting new memory every

 ISBN: 978-1-61208-367-4

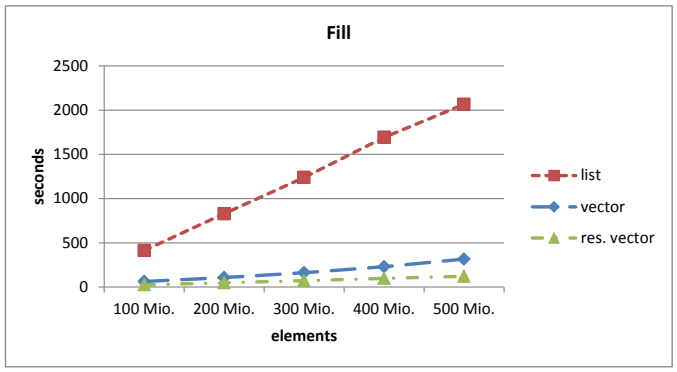


Figure 2. Filling data structures: The elements are inserted at the back. The res. vector gets the final size communicated via vector::reserve.

time. But if the vector is stored in a too small memory area, the entire container must be copied into a larger storage area, because of the contiguous memory order. Of course, the reserved vector does not have this problem, because it gets the final size at the very beginning.

For the list, it does not matter where the elements are stored in the memory. Because it is a doubly linked list with a tail pointer, it has also direct access to the last element. However, due to the link pointers, much more memory must be requested, and also the correct connectivity of the list nodes must be ensured. Much more salient is that the list must request for each node separately new memory. There must be permanently found free space on the heap, which represents a considerable overhead. This explains why the vector dominates the list.

B. *Insert sorted*

In this test case, both containers insert the same random numbers in sorted order. Before a new number can be inserted, the insertion position must be found. The range of these numbers moves between zero to 10,000. At the beginning, both data structures are empty. Figure 3 shows the results with different container sizes for vector and list.

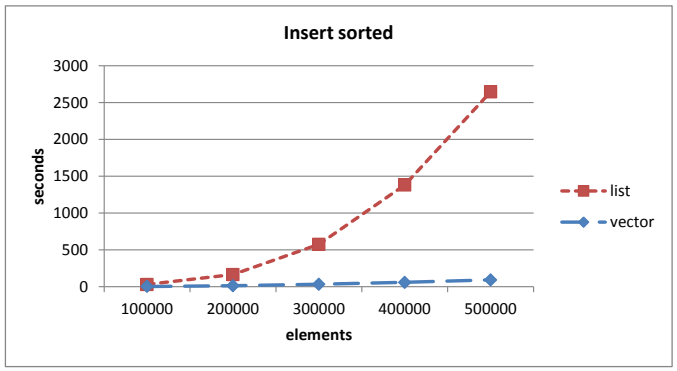


Figure 3. Insert sorted: list and vector in each case insert the same random numbers in sorted order.

It can be seen that the vector dominates this comparison very clearly. Because of its direct access to every element, the vector could use a binary search algorithm to find in $\mathcal{O}(log(n))$ the insertion position. However, the list must be run through linearly element by element. But this is not the only reason for the big advantage of the vector. Figure 4 shows a vector comparison, once with linear and once with binary search algorithm. The binary search algorithm is of course faster, but the difference is not large enough to justify the enormous advantage over the list.

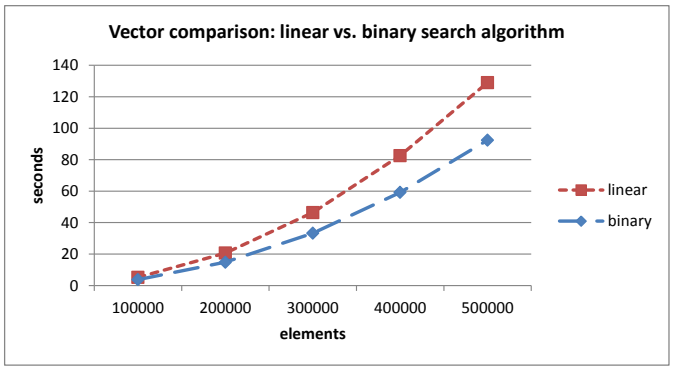


Figure 4. Vector comparison linear vs. binary search algorithm: The vector is with both algorithms much faster than the list.

Since the actual insertion of elements in a list takes hardly any time, finding the insertion position and allocating a new list node are responsible for the high run-time of the list. When the list must be linearly traversed, the address of the successor node can only be determined if the current list node is already loaded from the memory. Because of this data dependency, neither the compiler, nor the CPU has any opportunities to optimize. This explains the poor performance of the list.

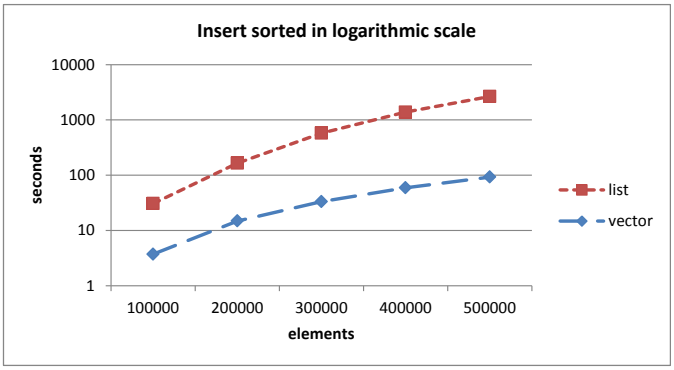


Figure 5. Insert sorted in logarithmic scale: In addition to the better run-time, also the better run-time behavior of the vector can be seen.

On the other side, the vector can be very well optimized thanks to its well-defined data order in the memory. It must be shifted on average half of the container on every insertion, but this takes in relation significantly less time than the list requires finding the insertion position and allocating a new list node. Figure 5 shows the same results as in Figure 3 in logarithmic scale. The vector needs $\mathcal{O}(log(n))$ to find the insertion position and $\mathcal{O}(n)$ to insert elements within the container. Although the list find the insertion position in $\mathcal{O}(n)$ and insert elements within the container in $\mathcal{O}(1)$, the vector has not only the better run-time, but also the the better run-time behavior.

 ISBN: 978-1-61208-367-4

C. Stable sort

In this test case, it is measured 1,000 times to sort the data structures. Both containers will be filled with the same random numbers in the range of zero to 10,000. Since list::sort is a stable function [13], for a fair comparison std::stable_sort is used for the vector. Both functions guarantee a time complexity of $\mathcal{O}(n * log(n))$ [14]. Figure 6 shows the results of this comparison.

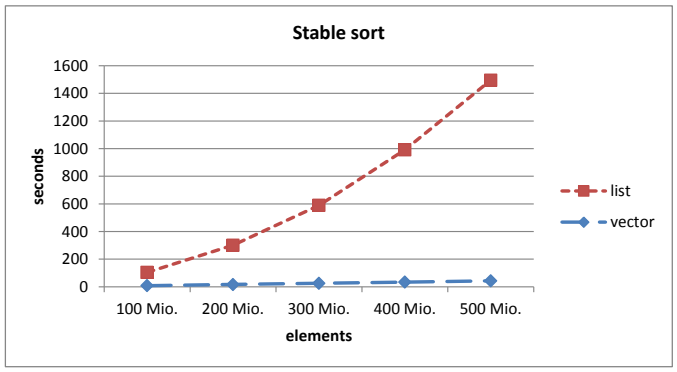


Figure 6. Stable sort: The same random numbers are sorted stable. It is clearly seen that the vector is in advantage, thanks to direct access and compact data.

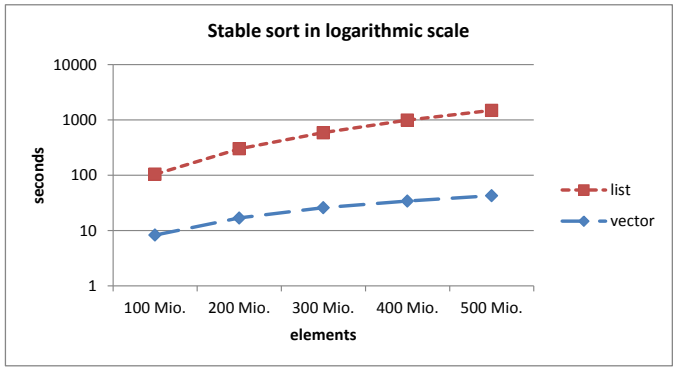


Figure 7. Stable sort in logarithmic scale: despite identical time complexity, the better run-time behavior of the vector can be seen.

Again, the vector dominates this test relatively clear. The direct access on each element in $\mathcal{O}(1)$ and the much more compact data structure of the vector have a positive effect again. Although both sorting algorithms guarantee a time complexity of $\mathcal{O}(n * log(n))$, the vector also has a better run-time behavior, as shown in Figure 7. The $\mathcal{O}$-notation only represents the worst case for the growth of the running time, which does not have to occur.

V. Evaluation

On the basis of theoretical considerations of vector and list, most people would think that it depends on the specific test case which container dominates which test. The run-time complexity of the data structures is quite equal in most of the used tests. But as the results show, a significant advantage of the vector can be seen. The vector dominates all tests with a single exception: ”Insert front”. There, the list has a clear advantage, because no element must be shifted. The smallest differences in run-time and run-time behavior show the test cases ”Insert Middle”, ”Is sorted” and ”Calculation of the arithmetic mean”. But even there the vector is at least twice as fast as the list, thanks to the direct access (”Insert Middle”) and the faster linear traversing (other two test cases). In the test cases ”Double each element”, ”Delete all occurrences of a number” and ”Delete all Duplicates” it is important to consider the characteristics of the vector. Linear traversing and the following insertion or deletion leads to unnecessarily high running times. But thanks to the direct access, elements could be doubled or deleted within the container with very less shift operations, which leads to a significant domination of the vector over the list. At the test cases ”Clearing data structures” and ”Reversing data structures” the vector is approximately ten times faster than the list. This is possible because of the comprehensible storage area of the vector, all elements can be cleared at once and the elements within the container can be easily swapped. The run-time behavior is quite equal in ”Reversing data structures” but at ”Clearing data structures” the vector has also the better run-time behavior.

But, what are the reasons for this clear result under the given test conditions? In this section, the main rules are filtered out from the amount of data that can be held responsible for these results. We have worked out four performance patterns representing some guidelines for todays software development on modern microprocessors.

1) **Linear traversing the data structures:** It is found across all tests that linear traversing of the data structures leads to significant differences in the duration time between vector and list. The vector benefits from the consecutive order of the elements in memory. This leads firstly to a maximum utilization of the limited caches. On the other hand it can be very well optimized, for example when elements are loaded speculatively already in advance. Such optimizations are becoming increasingly important, because main memory is becoming more and more slowly in relation to processor performance.
The elements of the list are scattered in memory, so they must be found costly and the memory access is poorly predictable. The address of the successor node can only be determined, if the current node has been read from the memory. Through this data dependency, the access to a list node can hardly be optimized, pre-loading data from memory (prefetching) is often impossible. Therefore, it takes far too long to go through a list.

2) **Access to items:** Often, a data structure should not be linear iterated, instead it must be random accessed on particular elements. The vector benefits from being able to access any element in constant very short time. This benefit allows to more than compensating other disadvantages. For example, when elements within a data structure should be inserted, deleted or swapped. The vector could use a binary search algorithm to find a specific element within a sorted container and also benefits from his direct access at sorting or reversing the container. The list does not have this advantage; direct access is only at the beginning and the end of the list possible. For any other element, the list must be

 ISBN: 978-1-61208-367-4

linearly traversed starting from the ends - with the same problems as shown previously.

3) **Inserting and deleting elements:** The insertion and deletion of elements at different positions of the data structures shows an ambivalent picture. At the end of the container the vector allows a very fast insertion and deletion of elements. The vector allocates memory always for multiple elements, to avoid requesting new memory every time. If the vector is stored in a too small memory area, the entire container must be copied into a larger storage area, because of the contiguous memory order. It is also possible to tell the vector the final container size at the very beginning, so all memory is requested directly whereby elements can be inserted even faster. When elements should be deleted at the end, the vector needs not to free any memory or iterate through the container, instead the elements can simply be cutted off, without changing the capacity of the container.
The list, however, must be run through in any case, with the exception of the first and the last node. Also, every time a new element is inserted or deleted, the list must request new memory or free used memory on the heap, which represents a considerable overhead. In addition, because of the administrative data, significantly more memory is used and the correct connectivity of the list nodes has to be ensured. Therefore the list needs significantly more time at the end of the container for such operations.
But the farther away from the end of the container elements should be inserted or deleted, the sooner the list has the advantage. The structure (interlinking) of the list has to be changed only locally. The vector must move a substantial number of elements when inserting or deleting, depending on the distance to the end of the container. This is an expensive operation. Therefore, insertion or deletion at the beginning of the vector should be avoided if possible. Usually this problem could be avoided or at least alleviated by clever optimizations or simply by using std::deque.

4) **Memory and cache utilization:** An aspect that should also be addressed is the high memory consumption of the list. The random order of the list nodes in memory leads to a high share of administrative data. At least two pointers need to be stored per element in a doubly linked list. What leads to 16 bytes per element management data in a 64-bit application. If there is not enough main memory available, it would lead to a significant drop in performance, because data must be outsourced on the hard drive.
Beyond that, this large administrative data make the limited caches ineffective, because more overhead and less useful data are stored. The random order of the list nodes leads also to a high number of load operations from main memory, because in the worst case every list node must be loaded separately. On the other side the vector requires very little administrative data and therefore allows storing data very compact. This allows utilizing the main memory and the limited caches best. And because of the contiguous storage area, multiple vector elements can be loaded at once from the main memory.

VI. Conclusion and Future Work

Summarizing, it can be said that modern microprocessors show a quite different run-time behavior for certain operations than one would expect looking at the corresponding $\mathcal{O}$-notation. In the mentioned cases, it is simply misleading following the $\mathcal{O}$-notation, which finally results in low performance of the application.

Certain optimizations of modern microprocessors prefer data structures with a coherent storage area. For lists, optimizations such as pre-fetching algorithms, are nearly useless, because of the data dependency of the link-pointer. The address of the successor node can only be determined if the current list node is already loaded from the memory, which makes it nearly impossible for modern microprocessor architectures to optimize the access to subsequent data. Against this background and the increasingly limited memory system, expensive memory accesses should be designed as predictable as possible and data should be kept compact to utilize the limited cache best. Such challenges prefer data structures with a coherent storage area.

It turns out that classical selection criteria for list and vector have been undermined by modern processor development in some way. So far, the insertion or deletion of elements within the amount of data was a clear domain of the list, since no shift operations are necessary. Today, even the shifting of the vector elements up to a specific position is more efficient than the linear iteration through the list to find the insertion position. If one takes into account certain characteristics of the vector, such data structures should clearly be preferred.

In the future, the performance of CPUs will still increase and additional potential for optimization will continue to prefer data structures with a comprehensible storage area. The so-called prefetching, the speculative load of data from memory, brings an enormous advantage for the vector. The data dependency of the list, which allows neither the CPU nor the compiler to optimize usefully, is a serious problem. This should be considered in software development.

Future work will evaluate additional data structures to get a more comprehensive picture about the run-time and the run-time behavior of different data structures on modern microprocessors.

References

[1] G. Pomberger and H. Dobler, Algorithms and Data Structures - A Systematic Introduction to Programming. Pearson Studium, 2008.

[2] D. E. Knuth, “Big omicron and big omega and big theta,” SIGACT News, vol. 8, no. 2, Apr. 1976, pp. 18–24.

[3] N. Wirth, Algorithms + Data Structures = Programs. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1978.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 3rd ed. MIT Press, 2009.

[5] D. R. Musser, G. J. Derge, and A. Saini, STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library, 2nd ed. Addison-Wesley, 2009.

 ISBN: 978-1-61208-367-4

[6] L. Liu and S. Rus, “Perflint: A context sensitive performance advisor for c++ programs,” in Proceedings of the 7th annual IEEE/ACM International Symposium on Code Generation and Optimization, Seattle, WA, 2009, pp. 265–274.

[7] C. Jung, “Effective techniques for understanding and improving data structure usage,” Ph.D. dissertation, Georgia Institute of Technology, 2013.

[8] G. E. Moore, “Cramming more components onto integrated circuits,” Proceedings of the IEEE, vol. 86, no. 1, 1998, pp. 82–85.

[9] G. E. Moore, “Progress in digital integrated electronics,” Electron Devices Meeting, 1975 International, vol. 21, 1975, pp. 11–13.

[10] J. L. Hennessy and D. A. Patterson, Computer Organization and Design - The Hardware / Software Interface, 5th ed. Morgan Kaufmann, 2013.

[11] J. L. Hennessy and D. A. Patterson, Computer Architecture - A Quantitative Approach, 5th ed. Morgan Kaufmann, 2011.

[12] A. Tanenbaum, Computer Architecture: Structures - Concepts - Basics, 5th ed. Pearson Studium, 2006.

[13] B. Stroustrup, The C++ Programming Language, 4th ed. Addison-Wesley, 2013.

[14] U. Breymann, C++ - Introduction and professional Programming, 9th ed. Carl Hansen, 2007.

 ISBN: 978-1-61208-367-4

Benchmarking the Performance of Hypervisors on Different Workloads

Devi Prasad Bhukya, Carlos Gonçalves, Diogo Gomes, Rui L. Aguiar
Instituto de Telecomunicações, Universidade de Aveiro, Aveiro, Portugal
bdeviprasad@av.it.pt, cgoncalves@av.it.pt, dgomes@av.it.pt, ruilaa@ua.pt

***Abstract*—Many organizations rely on a heterogeneous set of applications in virtual environment to deliver critical services to their customers. Different workloads utilize system resources at different levels. Depending on the resource utilization pattern some workloads may be better suited for hosting on a virtual platform. This paper discusses a framework for benchmarking the performance of Oracle database workloads, such as Online Analytical Processing (OLAP), Online Transaction Processing (OLTP), Web load and Email on two hypervisors, namely Xen and VMware. Further, Design of Experiments (DOE) is used to identify the significance of input parameters, and their overall effect on two hypervisors, which provides a quantitative and qualitative comparative analysis to customers with high degree of accuracy to choose the right hypervisor for their workload in datacenters.**

Keywords—Virtualization; DOE; Full Factorial Design; Main Effect; Interaction Effect

I. INTRODUCTION

Virtualization [1][2][3] is a technology where user can run more than one operating systems on a single system side by side. Initially, computer hardware was designed to run a single operating system at a time with a single application. This leaves most machines vastly underutilized. Virtualization lets user to create more than one Virtual Machine (VM) on a single physical machine and run different operating systems in different VMs with multiple applications on the same physical computer. Each VM shares the resources of that one physical computer across multiple environments under the monitoring of a Virtual Machine Monitor (VMM) or Hypervisor [4].

Virtualization works by inserting a thin layer of software called hypervisor directly on the computer hardware or on a host operating system. The virtualization architecture [5] we are using to do this experiment is by directly inserting the hypervisor on the hardware, which is called Bare-Metal architecture. Virtualization creates a virtual version of an operating system, a server, a storage device or network resources. The areas where virtualization is used are mainly network virtualization, storage virtualization and server virtualization. Virtualization is the best option available today for maximum utilization of the system resources by sharing of application and database. This helps to reduce the number of servers, hardware devices in the data center, which not only reduces the infrastructure and maintenance costs but also, reduces power consumption. The major benefit of using a Bare-Metal architecture is the overhead of a layer can be avoided; smaller footprint of the underlying Operating System (OS) uses considerably less system resources thereby granting the hypervisor and its VMs access to more Central Processing Unit (CPU) cycles, available memory and storage space on the hardware platform.

There are different types of virtualization, for different approaches. User has a choice to select any type of virtualization depending on his application/workload need. The main categories are Storage virtualization, Hardware virtualization, Network virtualization and Server virtualization. There are three types of virtualization techniques [6] that are mainly used i.e., Full Virtualization, Para Virtualization and Hardware Emulation.

Different workloads utilize hypervisor resources at different levels and depending on the resource utilization pattern some workloads may be better suited for hosting for particular hypervisor. This study is intended to compare how different Oracle workloads, such as Online Analytical Processing (OLAP), Online Transaction Processing (OLTP), Web Load and Email applications perform on different hypervisor environments. The rest of the work is organized as follows: Section 2 discusses about state of the art. Section 3 explains experimental procedure. Section 4 presents benchmarking analysis. Section 5 discusses about related work and Section 6 presents the conclusions.

II. STATE OF THE ART

Actually, performance of any system depends on various system and application factors. Higher performance is achieved in any system by tuning its individual system factors. Optimal values for each system tunable factor should be obtained by conducting several experimental runs and it takes a long time and blocks valuable resources such as cost, manpower and time. In traditional approach, performance benchmarking analysts are not aware of the experimental designs and analysis techniques often reaching misleading conclusions due to the following mistakes, such as: variations caused by experimental errors are not taken into account; important system parameters are not controlled; effects of different factors are not isolated; simple One-Factor-At-a-Time(OFAT) [22]designs are used; interactions among various factors are ignored; and too many experiments are conducted.

In addition, traditional performance tuning and benchmarking of hypervisor systems continues to be a tedious and time-consuming job with respect to any workloads. Since the features of upcoming hypervisor products are so complex, benchmarking needs in depth knowledge of the product and its domain. In the real world, hypervisor customers usually comes with a benchmarking requirement for their product with their competitors to software service based company and they will always go for a cost effective way of benchmarking. They also identify systems having many parameters that require careful hand tuning to get good performance.

 ISBN: 978-1-61208-367-4

In this work, a DOE methodology is proposed to overcome the above drawbacks and misleading conclusions of traditional approach of a hypervisors performance tuning and benchmarking experimentation.

III. EXPERIMENTAL WORK

The goal is to compare two hypervisors performance in Oracle workloads. In this study, we are interested to know how the hypervisors factors interact with the oracle workload. We conducted the benchmarking of the hypervisor with test bed configuration, as shown in Figure 1. Table 1 represents the hardware and their configuration. To virtualize a system two types of hypervisors are used Xen and VMware. The two hypervisors are identified as their kernel configurations are the same.

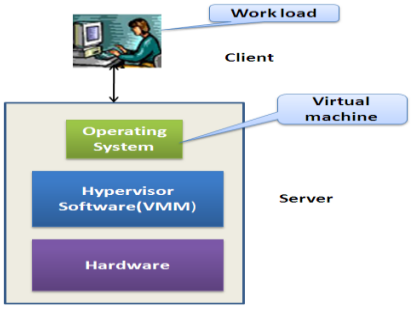


Figure 1. Architecture of virtualizing the system.

TABLE I. DESCRIPTION OF SYSTEM HARDWARE

Hardware	Configuration
Processor	Intel(R) Core(TM) 2 Duo CPU E7500 @ 2.93 GHz
NIC	Intel PRO/1000 PT
Physical memory	160 GB
RAM	4 GB

Since real database workloads are hard to obtain, benchmarks [7] were used as the database workloads in this research. Orion [8] is a tool for predicting the performance of an Oracle database without having to install Oracle or create a database. Orion is expressly designed for simulating Oracle database Input/output (I/O) workloads using the same I/O software stacks as in Oracle database. For each type of workload, Orion can run tests at different levels of I/O load to measure various performance metrics like Mbps, IOPS (Input Output per Second) and I/O latency. The Orion benchmarks developed by Oracle are widely accepted for testing the performance of Oracle database systems under various workloads. The Orion was installed in the VM and the application was pumped using Orion tool and its corresponding throughput (Mbps) recorded.

The performance of hypervisor depends on its various system factors. Each hypervisor have several factors associated with each other. Due to test lab availability and server limitation for this paper work, we have decided to take three important system factors of hypervisor with two levels and one factor with four levels at application level. These limitations are due to available servers in the present test lab. The vendors are set of two levels like Xen and VMware; Virtual CPUs (VCPU) with two levels like 1 GHz and 2GHz; Random Access Memory (RAM) with two levels like 1 GB and 3 GB, and Workloads are set of four levels i.e., OLTP, OLAP, Web load and E-mail. Table 2 summarizes the factors and levels.

TABLE II. DESCRIPTION OF FACTORS AND THEIR LEVELS IN THE EXPERIMENT

Factor	Level 1	Level 2	Level 3	Level 4
Vendor	VMware	Xen	-	-
Workloads	OLTP	OLAP	Web load	Email load
VCPU	1GHz	2GHz	-	-
RAM	1GB	3GB	-	-

TABLE III. DETAILS OF ORION

Application factors	Specification
No. Of disks	1
Disk Size	20GB
Run Type	Advanced
Time Taken	9 min (Random);15 min (Sequential)

In this paper, the workloads are simulated by using an Orion benchmark tool with their individual read write ratio, application block size and workload type. Table 3 represents the specifications of Orion while simulating the workloads in hypervisor benchmarking. Further, benchmarking of hypervisors for various real world workloads, such as OLAP, OLTP, Web load and E-mail workload have been proposed. The workload factors and their levels are summarized in Table 4. Table 5 describes the description of the VM system configuration. In this experiment, Windows Server 2003 guest operating system used. On the guest operating system the benchmark tool Orion version 10.2.0.1.0 were installed and VM configuration have been changed as per test case.

TABLE IV. DESCRIPTION OF THE WORKLOADS PROFILE

Workload	Read : Write Ratio	Application block size (kb)	Application type
OLTP	70:30	8	Random
OLAP	100:0	1024	Sequential
Web load	90:10	16	Random
Email load	50:50	4	Random

TABLE V. DETAILS VIRTUAL MACHINES

Virtual Machine	Description
VCPU	1 GHz,2 GHz
RAM	1 GB, 3 GB
Operating System	Windows Server 2003 Enterprise Edition (64-bit) with Service Pack 2
ORION	10.2.0.1.0

IV. BENCHMARKING ANALYSIS

To benchmark hypervisors over different workload, we used Design of Experiments (DOE), a robust statistical methodology, for optimizing the experiments. DOE [9][10] is used for performance tuning, optimization as well as screening the few vital factors to control the process. A process modeling method, DOE is defined as: A systematic and rigorous approach to engineering problem-solving that applies principles and techniques at the data collection stage so as to

 ISBN: 978-1-61208-367-4

ensure the generation of valid, defensible and supportable engineering conclusions. By using the interaction analysis features of DOE, one can benchmark their products. DOE can be applied for benchmarking framework and it has systematic procedure, as shown in Figure 2. In this study, benchmarks the hypervisor are also done in four steps: 1. objective of the experiment; 2. experimental design; 3. conducting experiment; and 4. data analysis.

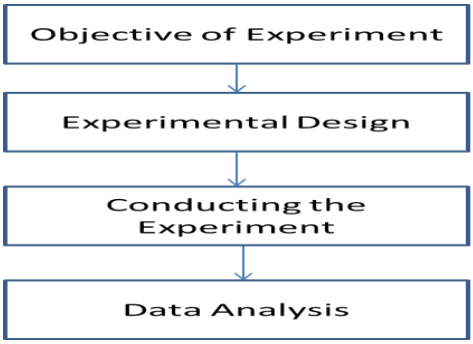


Figure 2. Flow chart of experiment.

Initially, we have to set the goal in the first stage of our experiment. Objective of the experiment plays an important role in any experiment as it is the area where experimenter identifies the problem and purpose of comparison; parameters of comparison; and also estimates the budget, schedule and resources for the experiment. The next step is experimental design where the experimenter will design the experiment like identifying the factors and levels, the design, the tools to be used in the experiment, etc. The design we choose in this paper is Full Factorial Design [11]. This design provides an option to conduct the runs of every combination without leaving any level of any factor. In this study, we have chosen four factors to benchmark the hypervisor and used statistical tool called Minitab [12] for DOE. The full factorial design has generated 96 test runs for this experiment. While generating test cases, the experimenter has the option of selecting techniques like replication, randomization and blocking in the experiment to reduce the experimental errors. The next stage is to conduct the experiment. In this stage the experimenter runs each test case and measuring the output using Orion tool. The experiment was carried out as per test case and the outputs were measured in MBPS. Each run were noted in the Minitab tool. Before analyzing the performance benchmarking of application workload, we need to check whether the response data is statistically sound or not in Minitab tool. The response data was tested with some prime data analyses namely, check for outlier [13], check for normality [14], check for any pattern and presence of time trend with the residuals of the response data [15]. In our experiment, we found that response data passes all above statistical test for further analyses.

A. *Main Effects Study*

The effect of the factor over the response is analyzed by using the main effect [16] feature of the DOE as shown in Figure 3 and it shows that performance of VMware is better when compared to Xen; performance of OLAP is better compared to the remaining chosen workloads, such as OLTP, Web load and email; performance of VCPU 1 is better than VCPU2; and performance of RAM with 1 GB has better performance when compared to RAM with 3 GB.

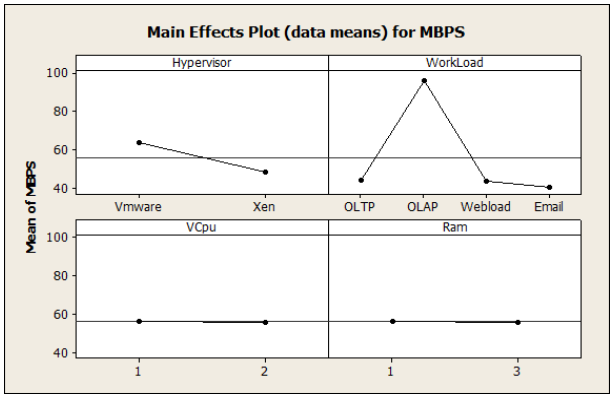


Figure 3. Main effects of factors vendors, workloads, VCPUs and RAM.

B. *Interaction Effect*

The combined effect of two or more than two different factors can be shown through DOE interaction effect [17]. The presence of interaction effect gives an idea about the impact of one factor on the level of the other factor. Figure 4 shows the interaction effect of hypervisors with workload over its throughput. It clearly reveals that VMware has better performance than Xen on most of the workloads. Eventually, the performance of OLTP workload is the same in both the hypervisors.

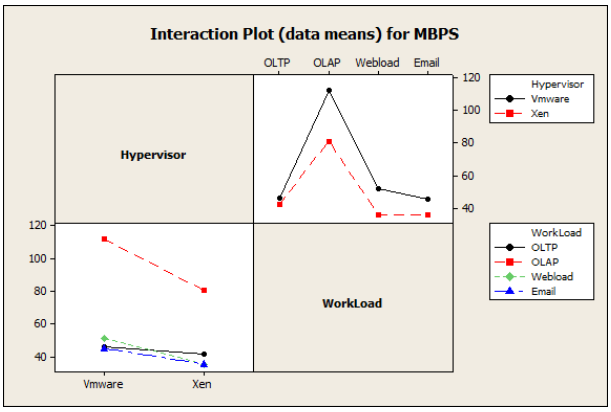


Figure 4. Interaction effects of hypervisors and workloads.

Figure 5 shows the interaction effect of hypervisors and VCPU. It shows that the performance of VMware has better performance than Xen. Further, the performance of VMware is better at 2GHz VCPU than 1GHz VCPU by 0.1633%. The performance of Xen is better at 1GHz VCPU than 2GHz VCPU by 1.7539%. The overall performance of VMware is better compare to Xen while using VCPU resources in VM.

 ISBN: 978-1-61208-367-4

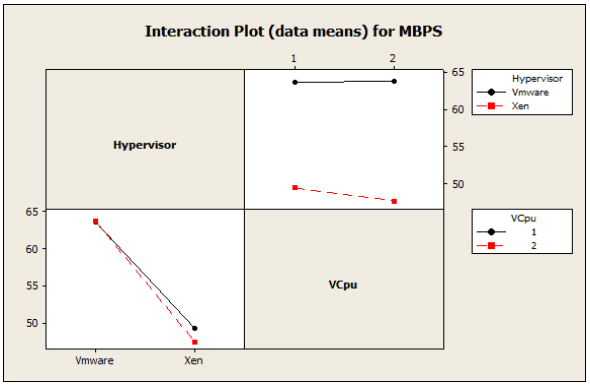


Figure 5. Interaction effect of hypervisors and VCPUs.

Figure 6 shows the interaction of hypervisors and RAM. It gives the information that the performance of VMware has better performance than Xen on both the RAM size. Although, the performance of VMware is better at 3GB RAM than 1GB RAM by 2.1991%. However, the performance of Xen is better at 1GB RAM than 3GB RAM by 3.9221%.

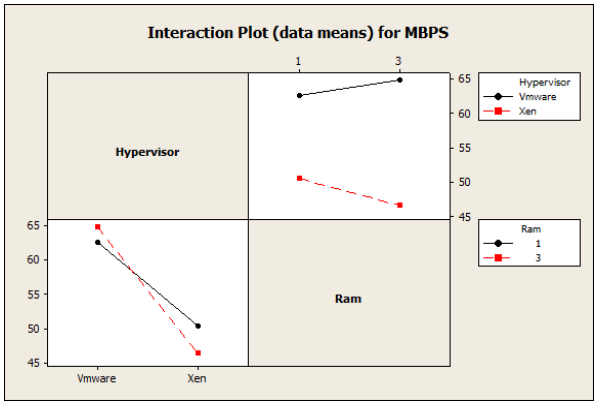


Figure 6. Interaction effects of hypervisor and RAM.

V. RELATED WORK

Many researchers have considered the performance of hypervisors but few of them focused on benchmarking the hypervisor technologies with workloads. The following papers are analysed to understand about the hypervisor performance and benchmarking in the present work.

Vijay Vardhan Reddy [18] conducts different performance tests on three hypervisors namely ESXi, Xenserver and KVM under private cloud environment. The author gathered the performance metrics of the mentioned hypervisors with various benchmarks tools such as Passmark, SIGAR and Netperf. The output of the paper was indicated that ESXi server and XEN server shows impressive performance in compare to KVM.

Andrea and Riccardo [19] investigated a quantitative comparison between Xen and KVM hypervisors. Their experiments shows that CPU performance provided by Xen hypervisor is best compare to KVM while increasing number of virtual machines runs concurrently on host which was using para-virtualized approach.

Walters et al. [20] evaluated the performance of Xen, OpenVZ, VMware Server for High Performance Computing (HPC) using OpenMP and MPI benchmark. In their experiments, it found that VMware server had worst performance compare to other hypervisors in most of the cases. In this work, they focused on HPC application performance.

Xavier et al. [21] studied the performance evaluation of container based virtualization technologies for HPC environments. They evaluated trade-off between Linux vServer, OpenVZ and Linux Containers for HPC systems. The results show that container based virtualization as alternative to HPC context when performance and isolation are required.

We believe our work is matching to the work presented in this section. We presented a framework for benchmarking the performance of Oracle database workloads, such as Online Analytical Processing (OLAP), Online Transaction Processing (OLTP), Web load and Email on two different hypervisors. We used DOE framework in our work to identify the key interaction among hypervisors and workload input factors with respect to their output. The output of the research paper may help the end users, IT industries to select the suitable hypervisor and workloads for their datacentre infrastructure requirements

VI. CONCLUSION

In this work, we have used the DOE for benchmarking the hypervisor on different oracle workloads. The outcome of the study shows that VMware outperform when compared to Xen. The experimental evaluation of hypervisors performance was done effectively by using DOE methodology. The investigating phenomena suggest that virtualizing their datacenters with Oracle database having maximum application on OLAP workload are recommended to choose VMware over to Xen. This approach also provides good product comparative analysis to customers with high degree of accuracy with good predictabilities in product improvement, product marketing and product selection.

ACKNOWLEDGEMENT

This work was supported by project Cloud Thinking (CENTRO-07-ST24-FEDER-002031), co-funded by QREN, “Mais Centro” program.

REFERENCES

[1] Smith, James E. "A unified view of virtualization." In Proceedings of the 1st ACM/USENIX international Conference on Virtual Execution Environments, pp. 1-1. ACM, 2005.

[2] Kroeker, Kirk L. "The evolution of virtualization." Communications of the ACM, pp. 18-20,52, no. 3,2009.

[3] Devi Prasad Bhukya, and S Ramachandram, “Performance Evaluation of Virtualization and Non Virtualization on Different Workloads using DOE Methodology”, IACSIT International Journal of Engineering and Technology, pp 404-408, Vol.1,No.5, 2009.

[4] Chen, Siming, Mingfa Zhu, and Limin Xiao. "Implementation of virtual time system for the distributed virtual machine monitor." In Computing, Communication, Control, and Management, 2009. CCCM 2009. ISECS International Colloquium on, pp. 571-576, vol. 4, IEEE, 2009.

[5] Wang, Jiang, Sameer Niphadkar, Angelos Stavrou, and Anup K. Ghosh. "A virtualization architecture for in-depth Kernel

 ISBN: 978-1-61208-367-4

isolation." In System Sciences (HICSS), 2010 43rd Hawaii International Conference on, pp. 1-10, IEEE, 2010.

[6] http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf

[7] Devi Prasad Bhukya, and Ramachandram S. "A Case Study of Identifying Benchmarking the Relative Performance of SAN Switches" – Proceedings of the International Conference on Software Engineering Theory and Practice (SETP-09), pp 120-127, Orlando, USA, July 13-16, 2009.

[8] www.oracle.com/technology/software/tech/orion/index.html

[9] D. C. Montgomery, "The use of Statistical Process Control and Design of Experiments in Product and Process Improvement", IEE Transactions, 24, 5, pp. 4-17, 1992.

[10] Antony, Jiju, and Frenie Jiju Antony. "Teaching advanced statistical techniques to industrial engineers and business managers." Journal of Engineering Design pp 89-100, 9, no. 1, 1998.

[11] Mutnury, Bhyrav, Nam Pham, Moises Cases, Daniel N. De Araujo, and Greg Pitner. "Design issues in telecommunication blade systems and their resolution through design of experiments." In Electronic Components and Technology Conference, 2009. ECTC 2009. 59th, pp. 1887-1893. IEEE, 2009.

[12] MinitabInc, http://www.minitab.com/products/minitab/

[13] Belsley, David A., Edwin Kuh, and Roy E. Welsch. Regression diagnostics: Identifying influential data and sources of collinearity. Vol. 571. John Wiley & Sons, 2005.

[14] Montgomery, and Douglas C. Design and analysis of experiments. John Wiley & Sons, 2008.

[15] Mason, Robert L., Richard F. Gunst, and James L. Hess. Statistical design and analysis of experiments: with applications to engineering and science. vol. 474. John Wiley & Sons, 2003.

[16] Keppel, Geoffrey. Design and analysis: A researcher's handbook . Prentice-Hall, Inc, 1991.

[17] Federer, Walter Theodore, "Experimental design", Experimental design. 1955.

[18] Reddy, P. Vijaya Vardhan, and Lakshmi Rajamani. "Evaluation of Different Hypervisors Performance in the Private Cloud with SIGAR Framework.", International Journal of Advanced Computer Science and Applications, 2014, pp. 60-66, vol.5, no.2

[19] Chierici, Andrea, and Riccardo Veraldi. "A quantitative comparison between xen and kvm." Journal of Physics: Conference Series., 2010, vol. 219. no. 4, IOP Publishing

[20] Walters, Chaudhary, Cha, Guercio Jr., and Gallo, "A comparison of virtualization technologies for HPC", IEEE International Conference."Advanced Information Networking and Applications, 2008.

[21] Xavier, Marcelo, Fabio, Tiago, Timoteo, Cesar , "Performance evaluation of container-based virtualization for high performance computing environments", IEEE Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2013

[22] Czitrom, Veronica. "One-factor-at-a-time versus designed experiments." The American Statistician, 1999, 126-131

 ISBN: 978-1-61208-367-4

Performance Engineering Using Performance Antipatterns in Distributed Systems

Chia-En Lin and Krishna Kavi
Department of Computer Science and Engineering
Unitersity of North Texas
Denton, TX USA
chiaen@unt.edu, Krishna.Kavi@unt.edu

***Abstract*—Performance analysis of software systems is complex due to the number of components and the interactions among them. Without the knowledge of experienced experts, it is futile to diagnose the performance anomaly and attempt to pinpoint the root causes in the system. Design patterns are a formal way of documenting best practice approaches in software development and system architecture design. Software performance antipatterns are similar to design patterns in that they indicate what to avoid and how to fix performance problems when they appear. Although the idea of applying antipatterns is promising, there are gaps in matching the symptoms and generating feedback solutions for redesign. In this work, we analyze performance antipatterns to extract detectable features, influential factors, and resource involvements so that we can lay the foundation to detect their presence. We propose a system abstraction layering model and suggestive profiling methods as the infrastructure in building the framework for performance antipattern detection with solution suggestions. It is used in the refactoring phase of the performance modeling process, and is synchronized with the software development life cycles. Proposed tools and utilities are implemented and have been used on real production servers with RUBiS benchmark.**

Keywords–Performance Engineering; Anomaly Detection; Performance Antipattern; Profiling.

I. INTRODUCTION

Developing a software system that meets its specifications demands continuous verification and validation efforts in iterative development cycles running from analysis and design, to implementation and deployment. During these processes, engineers build the system by creating design plans, and maintaining expected functional and non-functional properties of the specifications. Testing and debugging activities take place alongside the development. Similar to conventional functional debugging, non-functional properties must also be tested and appropriate fixes be made to meet the requirements. The complexity of modern software systems makes it difficult for the designer to assure compliance of non-functional requirements.

Design patterns are a formal way of documenting best practices in software development and system architecture design. The documented solutions are represented in a pattern language, which addresses a description of the solution to the problem, and the benefit gained from applying the process [1].

Since the usability of design patterns is still fairly abstract in terms of pattern matching, they are not easily adapted and applied in practice. Computing environment or context can be thought of as a multidimensional attribute set which has great impact on the execution of applications and systems. To better match the problem description, a design pattern has to provide specific context information for which the design pattern is intended. In most cases, the context provides detailed descriptions in a natural language to identify the scenario where the pattern is applicable and not applicable. Designers can look up the patterns and see if the scenario matches. Once found, they can apply the solution as a best practice to assure the result of the design is in fact the best possible.

Although patterns are promising and of great help in system development, some gaps between practice and application still exist. One of the obstacles results from the process of identifying the exact context and matching the scenario to the system under design. The context description of a design pattern is usually described informally in natural language; it is usually the responsibility of experienced domain experts to decide if the match is effective. For a relatively large-scale system, the complexity increases quickly making design patterns unusable.

Software design patterns can be thorough in treating functional design problems, but they do not address other aspects of the design. This leads to another gap in applying design patterns. Although the solutions to the design problems optimize the components of the system while building, they do not give clues to the quality of the design. In other words, only functional enhancements are ensured, whereas non-functional properties such as availability and reliability are not fully covered.

Smith et el. [2] are among the early proponents of performance design patterns. Principles of performance-oriented design are used as strategies in the development life cycle. They are embedded during the fundamental design practice which is later documented as performance patterns. Although performance patterns proposed are to address the performance issues, they are presented at higher levels, while the context can only be determined after the implementation of the pattern has been chosen.

Instead of following the same format of design patterns, the performance patterns are published from a different perspective, documenting potential bad practices that lead to poor performance. They tell us what not to do and how to fix a problem when it appears. Such patterns are called antipatterns. Performance antipatterns are similar to design patterns in that they document recurring problems, but state the scenarios from the opposite side of best practices. If the scenarios match with a performance antipattern, the predictive outcome of performance can be poor. The solutions of how to avoid the pitfalls are documented as solution descriptions analogous to best practices. The advantage of adapting performance antipatterns over performance patterns in practice is that they are easier to apply and are clearly guided due

 ISBN: 978-1-61208-367-4

to the explicit coverage of the scenario description space. However, antipatterns inherit one obstacle that is common with design patterns. It is still not straightforward to apply and gain the benefits from the solution. In this work, the focus is on targeting performance antipatterns in software development, and proposing approaches and tools to make the pattern application process more performance aware.

A novel framework that assists in performance debugging of distributed software systems is described in this work. To alleviate the obstacles of applying performance antipatterns during the software development life cycles, real performance indices are made available in our framework. Real performance baselines can be established so that the performance of the designed system can be compared to discover performance deficiencies. With the established facilities, contexts of performance antipatterns can be documented with practical metrics. It will assist practitioners to match, detect, and apply performance antipatterns quantitatively. For each system or sub-component being evaluated, the framework creates profiles in what is called suggestive profiling. When used during the development life cycle, it provides a realistic means both for antipattern detection and suggested solutions during the refactoring phase of a performance debugging process. Information regarding the root causes of the detected performance problem can be used to assist the redesign efforts. An effective solution can be devised and used to eliminate the identified performance anomaly.

The main contributions of our work are (a) an analysis of performance antipattern for detectable features, influential factors, and resource involvements (b) the proposition of a system abstraction layering model and suggestive profiling methods as foundations for performance antipattern detections, root cause analysis, and redesign suggestions, and (c) a performance antipattern detection and solution suggestion framework to be used in the refactoring phase of a performance modeling process, synchronized with the software development life cycles.

The structure of the remainder of this article is as follows. Analysis of performance antipatterns and how they are used in the design processes are introduced in Section II. The framework to adapt performance antipatterns in system and software development is presented in Section III, with the description of innovative fundamental architectures and suggestive profiling tools. Section IV describes the proposed process of performance antipattern detection and solution refactoring using the framework. Section V illustrates implementations and setup of the framework with examples. Section VI describes works that are closely related to ours. Finally, Section VII provides conclusions about this work and future extensions.

II. PERFORMANCE ANTIPATTERN

A. Performance Antipattern Analysis

Performance antipatterns were originally described by Smith and Williams [3][4][5]. Similar to the format of design patterns, documentation of an antipattern consists of the name of the pattern, the problem it addresses, and the best solution to solve the problem.

The first step in applying antipatterns in the design process is to extract the problem description and the feasibility for detecting the pattern in real systems. Since the research community frequently refers to these as fundamental antipatterns, there are additional attributes that highlight their usage features. For example, to be able to detect the existence of performance antipatterns, values of performance indicators have to be acquired to decide whether a specific symptom exists. Some of these can be determined by just a single value, while others require multiple samples over time. The former can be categorized as Single Value (SV), and the latter as Multiple Value (MV) antipatterns. These annotated attributes of a performance antipattern are summarized as Detectable Features (DF). A detectable feature is the extraction derived from the problem description statements, which serve as the essential indicators of existence of the pattern.

To apply solutions to overcome performance antipattens, the problem description is interpreted to extract the forces seen in the pattern description. Associated forces are defined as Influential Factors (IF) extracted from each antipattern will be used as the clues to the root causes. Forces can be extended when new forces are discovered from new archives. In a general system and software development context, the factors include:

Design Design factor is concerned with software objects and how well they are established in the design and implementation. It often relates to the policy in the design of resource sharing and recycling, as well as the arrangements of processing steps. Different design approaches result in different computing behaviors, and performance outcomes.

Algorithm Algorithm factor is distinct from Design in the way that software components can apply different strategies to achieve the same computation goal. The designer can adapt a strategic approach for computing and use different structures to manage data. Different complexities of the algorithms lead to different execution times.

Configuration Software development usually leaves options for configurations to let the user fine tune the behavior of the application to fit the usage expectation. While systems are ready to run, different management policies with corresponding configuration options can lead to different performance behaviors.

Threading Multitasking has been one of the frequent models used in software systems to cope with the complexity of parallel and distributed environments. Thread, as an abstract execution unit, plays a key role in carrying out a task along with other peer threads. Individual thread behavior, thread coordination, and management policies play a significant role in the overall system performance.

B. Design Processes with Performance Antipatterns

In most systems, the debugging activities are continuous along with an iterative software development life cycle. Analogous to general debugging activities, the performance debugging activity should also be embedded in the development process and run concurrently with the development processes to ensure the expected performance is on the right track. Taken from a generic modeling process, life cycle phases are put in order from requirement analysis and design to implementation and deployment testing. The life cycle is always iterative to make incremental improvements for each round. During the development process, engineers extract the required information to create models that assist in analyzing the design and planning for further verification and testing. These models are related to functionality of the system, and are used for validation and verification debugging purposes. Performance

 ISBN: 978-1-61208-367-4

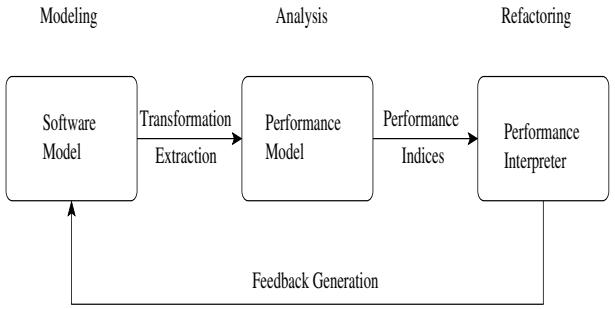


Figure 1. Performance Modeling Life Cycle with Performance Antipattern Refactoring

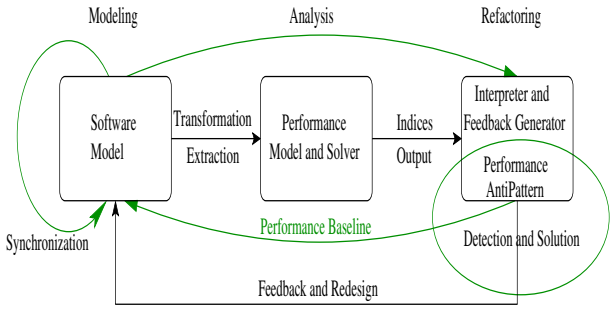


Figure 2. Refined Performance Modeling Life Cycle with Performance Antipattern Refactoring

debugging, on the other hand, requires further information to facilitate performance analysis and evaluation. In software performance engineering [6], a model-based performance analysis approach is adopted to generate performance data. They are created with the information about the architecture of the system, the capacity of its components, and the expected behavior of the system. Additional estimations such as request types and potential workloads are also needed for the modeling. The derived performance models are then used to produce quantitative numbers such as time duration, system utilization, and throughput. These values serve as indicators formally known as performance indices. The combination of these indices is used to predict the performance of the system.

The performance modeling process is depicted in Figure 1. The process is split into three phases. The modeling phase is the main stage of the system and software life span. Regular software models are built by following the life cycle phases. This software modeling phase is overlapped with performance modeling, because the updated performance attributes are gathered from the modeling activity as soon as the latest design revision is available. The second phase is analysis, and its goal is to create corresponding models for performance analysis and prediction. In this phase, model-to-model transformation is taking place. System and software models are transformed into performance models with information such as designated architecture, its topological layout, and available resources. In the analysis process, performance indices are obtained by solving the performance models using queueing network tools. These indices are used as indicators to forecast performance. Performance indices are interpreted in the third phase called refactoring. The goal of refactoring is to reflect the latest performance attributes and determine the satisfaction of the design in terms of performance qualification. If the performance does not meet the requirement, feedback can be generated to initiate design changes according to the interpreted results to resolve the performance issues. Engineers are obliged to check the predictive performance indices, and respond accordingly with changes to ensure the performance is acceptable. Analogous to software life cycles, the performance modeling process should proceed iteratively in an incremental order to synchronize with the original software model, create and analyze performance models to generate up-to-date performance indices, and give feedback with design changes for better performance.

In the performance engineering process, the goal is to detect a performance anomaly in the design and resolve the issue effectively and precisely. However, performance indices can only provide the location of the problematic components anomaly. To be able to come up with a change of plan, the practitioner must look into the design of the system to find the cause and estimate the performance penalties accordingly. It is difficult for performance experts to reason using only performance indices if the system being built is relatively new. This is where performance antipattern can help; especially in the refactoring phase. Running parallel with interpretation steps, an antipattern detection engine can be installed to assist performance antipattern identification. Once detected, the known solutions can be provided as feedback suggestions to remodel the system.

Antipattern detection mechanisms largely depend on the problem description to discover instances of a performance anomaly in the system, while feedback adjustments depend on the solution description. Ideally, the performance antipattern mechanism should be easy to adapt and build an engine for detection and find a solution. Figure 2 depicts the integrated process of the software development modeling and performance modeling processes. Both of the processes are synchronized in the modeling phase. In the refactoring phase, mechanisms of antipattern detection are integrated to assist in identifying performance problems and generate solutions accordingly as feedback for redesign. The integrated process is synchronized incrementally and iteratively with the software modeling.

Although the promises of performance antipatterns, or design patterns in general, are great, the intricate nature of documenting a scenario and its environment in a computing system makes direct application of antipatterns difficult. If the goal is to put it into automatic practice, there are many gaps and challenges. One example of deficiency is the difficulty in recognizing the context where the performance antipattern exists. For instance, in *unbalanced processing-extensive processing* antipattern, the problem description states "the extensive processing impedes overall response time." It is left to the discretion of engineers to realize what exactly the response time change is and how it should be modeled in the specific application. Another example in the *ramp* antipattern, the statement like "processing time increases" in the problem description, has to be determined by the engineers as to what is the significance of time indices in modeling so as to detect the symptom in the specific application.

Another noticeable hurdle in applying performance antipatterns is getting the solutions as feedback. Inherited from generic design patterns, solution descriptions are essential parts in pattern documentation that carry key expert knowledge. The secret to completing the performance modeling process

 ISBN: 978-1-61208-367-4

TABLE I. SYSTEM ABSTRACTION LAYERS

Division	Layer	Design Abstraction
Software	System	
	Sub-System	Integration
	Component	Composition
	Task	Configuration
	Thread	Execution
Platform	Layer	Abstraction
	Middleware	Resource Management
	Operating System	Scheduling Policy
Hardware	Layer	Abstraction
	Execution Unit	Processing Element

largely depends on the precision of solutions, which enables debugging processes to tweak the design to overcome the performance pitfalls mentioned in the antipattern. The problem with context ambiguity, similar to the counterparts of performance antipatterns in detection the mechanism mentioned above, appears in the solution description as well. For example, in *more is less* antipattern, to determine whether the architecture can meet its performance goals by staying below the thresholds, one has to decide the appropriate value for the threshold. The threshold value is not only affected by the underlying architecture attributes; it can also be affected by the degree of discretion which in turn depends on the context of the application. Another example, in *Traffic Jam* antipattern, one of the solutions is to provide sufficient processing power to handle the worst-case load. The processing power adjustment is an open issue to be determined to remedy the bottleneck. These examples show that applying performance antipatterns in the refactoring phase would need reasoning tools to assist the feedback generation. In other words, once an instance of antipattern has been identified, applying the solution description to generate feedback for system improvement is not straightforward. Tools that can reason about the context for the specific system should be available to enable the reasoning process. This is where profiling approaches can be useful, which are described in the next sections.

III. PERFORMANCE EVALUATION FRAMEWORK

A. System Abstraction Layers

Since performance results derive from the integration and cooperation of software and system architecture, an abstract structure is proposed to help us identify essential elements of the performance forces and express how they organize and play different roles in computing. A system and all its entities is modeled in a structure called System Abstraction Layers (SAL). The modeled layers consist of three divisions from top to bottom: software, platform, and hardware. Table I depicts the contents of each layer and their design abstraction. Each design abstraction represents orthogonal forces from a system development activity that affect the behavior of the elements in the layer.

In the software layer, systems are realized by the integration of sub-systems, each of which is responsible for a specific functionality. For each subsystem at the software layer, entities of software in terms of components and libraries are composed to create the sub-system. Each component in this setting is executing the tasks designated. The abstraction of task control can be related to configuration if the tuning mechanism is available for the software entity. The real execution in the software layer to carry out the tasks of a component is given to the elementary execution entity known as the thread. The abstraction is also compatible with the implementation using only the processes, where each process is treated as a special main thread.

Many software systems need to take advantage of using services from middleware to ease the complexity of developing and deploying applications. Middleware is used to manage the communication resources and hide the interaction details from the users, especially for distributed systems. In a broader sense, it also manages the server resources by regulating how control and information flow is distributed under the designated architecture topology. Below the middleware, it is the operating system that provides services for resource management and process scheduling. The platform layer is about resource management where the system and its software entities reside and access the computing resources.

The bottom layer of the server under the platform layer is related to hardware component organization. It is where the actual performance is measured. Performance revealed from the hardware layer depends on the grade of components installed. Utilization of hardware components can be acquired from this layer which includes processors, memory, network, and disks among others.

With the defined conceptual layers in SAL, each of them relating to a specific design abstraction, we can describe a performance scenario flexibly both at higher and lower layers. A high level scenario expression can be refined and mapped to its corresponding lower level counterparts. Through the process of mapping, we can identify the related elements in each layer and reason about the forces associated with them. This lays out the foundation to detect the case of a performance anomaly, when the root cause elements can be identified in the hierarchical approach. To accommodate the context information, the structure of layers can be represented with a description language. The performance context of a system or application can likewise be expressed.

B. Performance Suggestive Profiling

The purpose of profiling in our framework is to identify the performance anomaly and to locate the root causes. Once performance antipatterns appear in the performance modeling, the practitioner should be able to detect and get the suggestive solutions depending on the current context of the system to remedy the problem. To gain the causality reasoning capability, the proposed profiling mechanism is set to conform to the system abstract layers. The profiling mechanism can also serve as the toolkit to access the performance baselines in the SAL structure. For the assistance role, the profiling mechanism should be compatible to both software development and the performance modeling process, making it easy to adapt in all phases in the process. The profiling mechanism should be easy to setup for performance testing and evaluation. Since profiling is also applied to the baseline, profiling can aid in detecting antipatterns and suggest solutions. With these design requirements in mind, the following discussion provides the design rationale and discusses the components of the profiling mechanism, the context where they can be applied, and how they can be utilized in the system development.

Conventional software profilers usually focus on the source code or its corresponding executable binaries to get statistical measurements of the software package or library. The infor-

 ISBN: 978-1-61208-367-4

TABLE II. SUGGESTIVE PROFILING METHOD IN THE SYSTEM ABSTRACT LAYER CONTEXT

SAL Context	Profiling Method	Suggestive Profiling Method
Subsystem, component	Path-Oriented	Alternative Path options
Thread	Thread Behavior	Thread Behavior comparisons
Middleware	Networking Profiling	Request traces and communication protocol verification
Hardware	System Resource Profiling	Physical and Abstract resource summarization

mation includes frequency and duration of routines such as function calls. The goal of a conventional profiler is program optimization. System profilers focus on resource usages of the server. They monitor the state of hardware resources such as processors, and report consumption summaries. All of these profiling mechanisms are essential to our purpose. However, more precision and reasoning structures are needed to achieve our goal. We need the following:

- Specific timeline information that can identify not only spatial hot spots but also the temporal features.
- Profiling information from one layered aspect that can relate to another, such that a reasonable mapping can be inferred.
- Profiling information that can be summarized and compared with the associated computing context.
- Inter and intra communication should be integrated in the profiling mechanism.

To this end, we put together the profiling mechanism needed to fulfill the requirements of our purpose. In particular, our goal is to assist in performance antipattern detection, as well as feedback generation. In addition, software systems often run in networked environments; the profiling mechanism needs to flexibly accommodate and adapt to distributed settings as well. In Table II, the profiling mechanisms are categorized into contexts that are matched to SALs. Each of the profiling methods is given a brief description followed by its suggestive profiling method. The purpose of the method is to explore other available options in the same context level of the system to give leeway in enhancing the performance result or avoiding bad practices. The practitioner can take advantage of the suggestive profiling approach to explore design options to achieve better performance. In the performance refactoring phase for antipattern detection, exploring the suggestive solution methods may provide a clue to a final solution.

1) Path Profiling: The framework adapted the terminology of Path Profiling from data flow analysis [7], and the pathwise decomposition concept from path-oriented analysis [8]. The concept of path-oriented profiling is based on the measurement of different execution paths. If we can make the most common path execute faster, the response time may be shorter. Path profiling also provides insights on improving performance by revising the chances of executing certain paths, or improving the efficiency of the path. If the frequency of execution of a path is relatively high, the savings in execution time can become significant. The dependency of paths and associated components can be identified.

Path profiling can be seen mostly at the level of software components and libraries, and in the programs. In the software layer, execution path is the lowest unit of refinement for the software system. For each thread, path profiling is also essential to discover performance problems. Software elements can be explored along the execution path.

The concept of execution path can be extended to accommodate information flows. Information flow tracking in the program is done to understand the pattern of execution paths as certain requests are being executed. It can also be extended to include communication routes that connect the execution path between server nodes. The high level view of path profiling can be observed at the subsystem level where interactions between clients and servers use different routes. Alternative routes between them may be the result of dispatching policy or adapting flexible algorithms to react to traffic congestions.

2) Thread Behavior Profiling: A thread is a sequence of instructions and the representation of a logical computational unit, which can be scheduled to run by the operating system. A pool of working threads can be initialized before the real workload picks up and be ready to respond without delay. Adapting this thread model also has the benefit of executing true concurrency in a multicore environment. Thread Behavior profiling is about the observation of thread creation, execution, destruction, and management of threads. At the system architecture level, processor affinity can be monitored as multithreaded programming specifies the arrangements. The combination of resource distribution and the management policy such as the number of threads and their running priorities affect the overall performance. Threads can also be viewed as another form of dynamic path, because every thread runs on its own copy of instructions. The observation and summary of individual threads can be performance indicators of how well they coordinate and cooperate.

The context of thread profiling is at the task level where the system adapts a multithreaded programming model to carry out designated services. Depending on the features of the application, a threading system usually provides facilities to adjust the behavior of threads to improve their performance. A thread is the lowest logical task unit that we can monitor in the profiling. It provides the flexibility of measurement in both higher and lower levels of the system. At the higher level, an end-to-end performance scenario can be profiled by integrating thread behavior profiling in each subsystem with the information flows. At the lower level, each task and its resource usage by a specific thread can be analyzed. The flexibility of thread monitoring facilitates the whole system profiling at a fine-grained level.

3) Network Profiling: Networked systems have become the infrastructure for every computing system no matter where it resides, either in enterprise clusters, virtual hosts, or Clouds. The complexity of interaction patterns among servers increases exponentially. As the network becomes the computing platform, it isnecessary to profile and monitor network traffic. Our model classified the networking in the context of middleware and includes proxy, router, programming middleware, and other network topologies such as multi-tier and clustering. Network profiling focuses on getting information about requests and responses, and the underlying communication protocols. Measurement information can be about the number of requests at the higher level, and the number of network packets at a lower level. It can also look into the data that packets carry, and profile the characteristics of the request message. Performance of the network activities can contribute to either the network interface capacity such as queueing buffers, or the processing speed of the server.

Network profiling information can be referenced by its connected systems and software components to make more

 ISBN: 978-1-61208-367-4

informed design decisions. It can also contribute to the construction of analytical models such as queueing networks with more realistic estimation of queue lengths and arrival rates. Finally, for information flow analysis, networking profiling facilitates visualization of the activities in detail, and provides the overall picture of the networked components and systems.

4) Resource Profiling: Conventional resource profiling is about profiling physical resource usage. The resource here refers to the hardware components of the server architecture. The higher capacity the server has, the higher the performance. In practice, cost, overhead, and other limitations can affect the choices and ability to obtain more capacity. Resource profiling gives information about resource characterization of the server and its computational capacity. It usually hints at the potential performance outcome of the system.

One puzzle of resource profiling is in the context of the software abstraction layer, where engineers often want to know precisely the resource consumption of a specific software entity. For example, if the load a working thread contributed to a processor load can be calculated, then the total resource requirements can be estimated given the number of threads. Physical resource monitoring alone cannot fulfill the job because it is hard to get a clear view of workload of a working thread without being affected by other programs and the operating system management policies.

To estimate the resource consumption of a software entity, instead of applying measurement and estimation using a system utility, we turn to resource consumption estimation using the number of instructions executed. Resource profiling in a software entity can be abstracted to the lowest level of computation using instructions before putting them into operation. These instructions are the ones that are consuming resources. Therefore, the system resources that need to accommodate these executions can be estimated. Depending on the type of resource, a processor's workload can be approximated by the number of instructions running on it. The instructions can be further categorized by the type of memory access that can have different number of cycles. For a storage disk, the number of I/O accesses resulting from the execution can also be counted toward resource consumption. Advanced resource modeling such as register and cache behavior can be devised to extract the measurements. We name this type of resource profiling as abstract resource profiling in contrast to the physical resource profiling method discussed above. The mapping between software entity and the abstract resource can be clearly identified and the resource consumption can be reasonably estimated.

The application of resource profiling can be applied to most of the system. Software systems usually adapt monitoring mechanisms to extract the information from system hardware components. Abstract resource profiling will need the other profiling mechanisms in the suggestive profiling to supply the measurement. All the conventional resources in the antipattern domain, such as CPU, Memory, Network, and Disk, can participate in resource profiling.

5) Request and Workload Profiling: Workload plays an influential role on the performance. Although the profiling method is not categorized into any system abstract layer, it affects every part of the system. Depending on request patterns, one subsystem or component may have a larger workload then that of the others. If the workload exceeds the planned capacity of components, a bottleneck would occur. Similar considerations apply to the performance impact of threads, in that they may spend more time processing requests, and the throughput may suffer. In the context of middleware, interactions between servers may take longer under heavy workload due to waiting for responses. In the system architecture layer, the computing workload coming from the above layers has a direct impact on resources and overheads when switching between tasks to meet the services.

The request and workload profiling provides information about the impact on each context of the system. Engineers can evaluate the scenario of request pattern in each focused context independently. The profile relating only to an individual context can be used as a specific source of information to focus on that particular design improvement. On the other hand, the profiler can characterize the workload, so that the design of the system can be adjusted flexibly if it is possible to increase the performance. For example, if the profile of the request is CPU bound, engineers may consider distributing them as evenly as possible to available servers. Another alternative solution occurs when CPU bound requests prefer to be sent to a CPU with higher performance. We also note that both physical and abstract resource profiling information can be used to characterize workloads. This information can help engineers understand the impact of requests.

In practice, request types and their temporal patterns are dynamic, and the workload characteristic is not known a priori. The suggestive profiling mechanism can be used at deployment to selectively monitor requests, and create the workload profile associated with the context. The profiling can select time periods or focus on a specific component for performance monitoring.

IV. Detection and Solution Suggestion Processes

To be able to use suggestive profiling in performance antipattern detection and solution feedback, one has to understand performance baseline. Performance baseline is the summary of current performance of the system. It can be used for performance debugging to check against requirements. Preliminary performance evaluation can be obtained by establishing the baseline of the target system or components. The content structure of the baseline is compatible with the system abstraction layer, in which path-oriented, threading, networking, and resource profiles are recorded. In each context of the profiling, performance metrics such as execution time and process utilization are available for verification. Performance baseline can be created for every element in each context of suggestive profiling method including subsystem, component, thread, network, and hardware component. Depending on the needs of debugging activity, engineers can zoom in on targeted components and their interactions when high level information is not enough. In short, performance baseline is an agile performance filtering and debugging tool used in the software development process to collect targeted performance snapshots.

With performance baseline as the debugging framework used in the system development process, activities in performance modeling processes can share the data it collects. Since both of the processes are synchronized, the performance metrics collected are reflected in the latest status of the system. In the refactoring phase, performance antipattern detection and solution suggestion feedback mechanism can be executed with the help of suggestive profiling. Performance baseline

 ISBN: 978-1-61208-367-4

is accessed to extract the metrics of the detectable features needed with each antipattern. The value of detectable features can be checked to see if the symptom appears. If not detected then the antipattern does not exit, and no action is needed. If detected, the solution may be applied to solve the performance anomaly in the system. Antipattern solution can point to problem spots and give approaches to resolve the problem. As discussed in previous sections, refactoring dilemma exists because we need more clues to generate detailed feedback for redesign and eliminate the anomaly. To close the gap, suggestive profiling can be used to narrow down the root causes.

We recall that the suggestive profiling method consists of profiling path, thread behavior, networking, and resources in the layered context defined in SAL, and each profile can be evaluated independently. In order to converge on to the root cause, we examine the suspicious context revealed by profiling. Within the root cause, performance metrics gathered by the profiling mechanism are verified against the performance antipattern symptoms to discover corresponding solutions. For example, if the root cause specifies a component is the bottleneck in the system, we can further analyze execution time profiles of paths, and make a specific solution suggestion in refactoring. Although we reason at the specific component level, levels higher and lower than the root cause context can also be inferred for potential redesign options. For example, if the root cause is a thread's performance, execution paths at a higher level or resources at a lower level can be inferred as the relevant factors. Solution suggestions can therefore provide more relevant information.

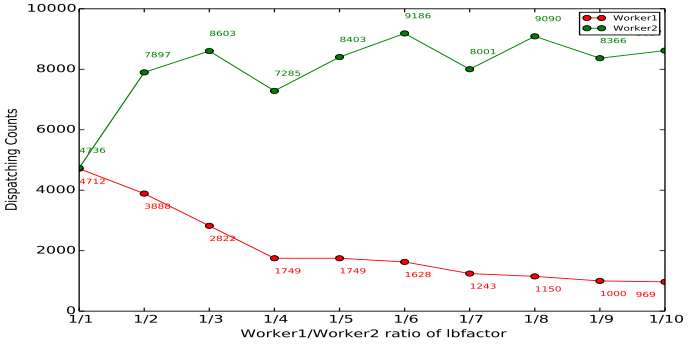


Figure 3. Number of Requests Dispatched to worker1 and worker2 of JBoss

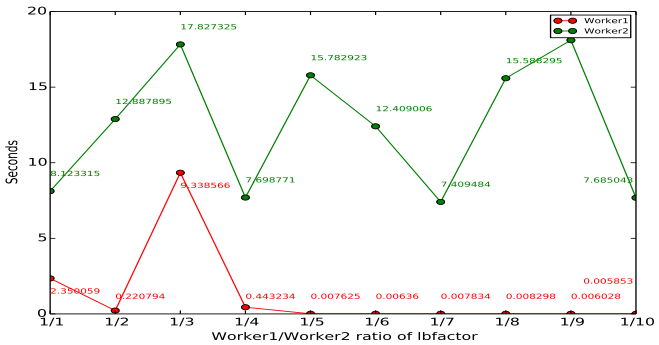


Figure 4. Average Execution Time of Requests Dispatched to worker1 and worker2 of JBoss

V. Case Study

A. Framework and Tools Implementations

The suggestive profiling was implemented using Pin tool [9]. An associated data analytic framework for performance debugging, antipattern detection, and solution suggestions were created to work with the tool. Together they can trace each executed instruction of an application for path-oriented analysis. It also provides other instrumentation points including basic blocks, routines, images, and complete application. These abstractions can be used to identify call graphs, accesses to libraries, and inter and intra component communication, which can easily fit in the system models.

In the framework, our tool includes following utilities to facilitate suggestive profiling:

- Data collection, processing, and management for different profiling methods.
- Communication between software components and systems is profiled with the help of protocol plugins. Each plugin specifies the pattern of interactions.
- System resource monitoring, logging, and analysis.

B. Experimental Setup in Production Systems

RUBiS [10] was setup on Xen 3.1.2 virtual machines hosted on Dell Optiplex 960 with 4 CPU and 4GB RAM. Each virtual machine runs on one virtual CPU and 512MB RAM. Each virtual server is connected to a virtual network interface with a unique network address. The virtual network connection is created by an ethernet bridge, and a DHCP server is setup to assign unique network address to each virtual server.

RUBiS was installed with Apache2 httpd [11], JBoss AS 4.3.2 [12], and MySQL [13] as the web server, application server, and database respectively. The suggestive profiling Pin tool was installed on the web server to generate the performance baseline for the web server, and to demonstrate performance antipattern detection and solution feedback suggestions. On each server virtual machine, we measure the load with sysstat utility to collect CPU, memory, network, and disk usage every one second. All the traces and logs generated from the suggestive profiling Pin tool, and the system utility measurement logs are collected afterward to avoid interferences with the workload of the server.

C. Performance Antipattern Detection and Solution Suggestion

The proposed framework is applied to detect root causes using the exemplified performance antipatterns in this study. Once a performance antipattern is documented and relevant context-dependent solution suggestions are recorded, they can be supplied directly as possible solutions. Practitioners have the option to choose between applying documented solutions, or creating a new antipattern instance that is specific to the scenario of the system under review. A short discussion for each antipattern studied in our case study are described below.

- Unbalanced Processing Antipattern

  Description Problem occurs when processing connot make use of available processors.

  Application Best practices of dispatching between an Apache Web Server(WS) and multiple JBoss Application Servers(ASs).

  Detection Unbalanced processor utilization or service

time duration observed between ASs.

Solution Adjust dispatching configurations by making changes to the proportion of load sent to different wrokers, using lbfactor in mod_jk.

Experiment Experiment using default transition workload with 500 users whose requests are served by two JBoss ASs and one Apache WS is presented with different ratios of lbfactors. Figure 3 depicts the number of requests dispatched from WS to either worker1 of AS1 or worker2 of AS2. The load balance ratios are marked as the ticks on x-axis. We observed that the number of requests to worker1 and worker2 are approximately proportional to the weight of the lbfactor. Figure 4 depicts the execution times of requests dispatched from WS to worker1 or worker2 with different dispatching ratios. The represenation is not liner, but it reflects the scenario in which the preferred node spends more time processing because of the improper load setting. Both the dispatching number and the time duration with different ratio are interrelated.

- More is Less Antipattern

Description Problem occurs when a system spends more time trashing than accomplishing real work because there are too many processes relative to available resources.

Application Best practices of deciding what is the appropriate number of working threads needed to serve in an Apache Web Server.

Detection Comparison of throughputs between settings using different number of threads.

Solution Adjust the number of threads based on peformance.

Experiment In this experiment, the RUBiS benchmark with different sets of configurations was run, and the outcome of the performance baselines and their differences were observed. Figure 5 depicts a test run with 800 users using various sets of worker configuration shown in the legend. The numbers in the legend correspond to the order of Apache httpd server's configuration variables: StartServers, MinSpareThreads, MaxSpareThreads, ThreadsPerChild, MaxRequestWorkers, and MaxConnectionPerChild. For each request types from the benchmark, the corresponding average response time in seconds is shown.

- God Class Antipattern

Description Problem occurs when a single class either performs all of the work or holds all of the data of the application.

Application Checking both design and implementations for better object-oriented paradigms.

Detection The number of control or data flow in a programming class that is higher than predefined threshold.

Solution Refactor the design and implementations of the detected class.

Experiment Developer documentation of httpd states that, all requests pass through ap_process_request_internal() in request.c of the web server. We want to observe the information flow and its frequencies when a real workload is used. Before checking the flow of information to the targeted function from suggestive profiling tool, symbol table of the ap_process_request_internal() function is extracted from the Executable and Linkable Format (ELF) of httpd to find the static address of the function. This information is used to acquire the structure of instructions in execution order. Request flow information collected from suggestive profiling tool is checked with the static structure of the function to produce the request flow graph at run time. It should be noted that this type of analysis is possible only when source code is available.

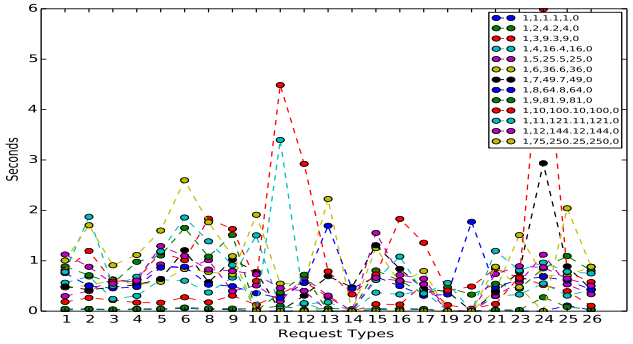


Figure 5. RUBiS Benchmark with 800 Users Using Various Sets of MPM Configuration

VI. Related Works

In the following subsections, approaches related to performance antipattern detection, diagnosis, and solutions that are closely related to our work are discussed.

A. Performance Antipattern Detection

Performance antipattern detection has been addressed in different systems and models. Performance detection in component based enterprise systems was proposed in [14], where a rule-based performance diagnostic tool is presented. The tool can work with EJB applications, in which data from runtime systems is extracted and applied with rules for antipattern detection. The method is limited to EJB systems. Another performance detection and solution approach presented in [15] discusses the performance antipattern in the context of the Palladio Component Model (PCM) [16] software architecture modeling language. A queueing model is derived from the software model in PCM, and is solved to generate performance indicators. The predictive values are matched against performance antipattern rules in PCM to determine whether an antipattern exists. Once detected, solutions can be applied. It uses iterative processes to solve antipattern one by one. A similar approach but using Architecture Description Language (ADL) can be found in [17]. In [18], performance antipatterns are presented using logical predicates. The problem description for an antipattern is interpreted and presented using first order logic equations. The approach focused on antipattern presentation and detection. In [19], Performance Problem Diagnostics (PPD) approach combines search techniques with systematic experiments for performance antipatterns detections. The search is based on a decision tree technique to locate possible root causes, while the detection strategies are based on goal-oriented experiments. All these techniques described

 ISBN: 978-1-61208-367-4

above are based on models and heuristics, and do not describe how to understand baseline performance or setting threshold values at which system configurations should be changed. Our framework relies on runtime data collection to understand performance bottlenecks and how to tune system parameters.

B. Automatic Diagnosis and Feedback Generation

In [20], a rule-based automatic software performance diagnosis framework is proposed for detecting performance bottlenecks. Layered queueing models are used to generate performance predictions. The generated performance indices are checked against predefined rules to detect performance bottlenecks. The rules will also suggest mitigation approaches to reduce operations or add resources. The solution feedback is largely dependent on the definition of the rules. The success of the system depends on the extensibility of the rules. The feedback solution depends on translating performance model attributes in design, which are not provided. A similar approach is presented in [21], which extracts software and system architecture and creates a queueing model for performance anomaly detection. In the feedback process, the architecture model is used for redesign considerations. Our framework does not use queueing modules, but relies on profiling. An approach is proposed to address these concerns regarding detection and solution feedback with real system thresholds so that performance antipatterns can be applied in real practice. In [22], a special detection approach for finding the most guilty performance antipattern is proposed. The process checks performance antipattern symptoms against system requirements, and filters out the ones that do not violate them. The final list of performance antipatterns are ranked using scores calculated from equations defined for specific performance criteria. In our current framework, we do not rank the antipatterns. However, the use of baseline will eliminate some antipatterns from consideration, if the performance is acceptable.

C. Solution Suggestions

There are many published approaches suggesting solutions to overcome performance bottlenecks. Our discussion here focuses on the ones that are related to performance anomaly detection and root cause analysis. In [23], the performance anomaly clustering method is used to narrow suspicious components in distributed systems. Clusters are used to chain components together when they are affected by the same root causes. The clustering is based on the similarity of the performance indices. To identify the problematic performance spots, relationships between groups of clusters are compared. Thus performance anomalies are identified at higher levels: such as the server level. Further diagnosis steps will need to rely on the practitioner's system knowledge. A framework for controlling system configuration parameters to adjust performance was proposed by Stewart et al. [24]. The coverage of the approach depends on the number of controlled configuration used. In practice, it is not feasible that every configuration and manifestation can be covered. Our approach collects data on the software and the system, and establishes the performance measurement specifically reflecting the real scenarios of the system under performance debugging. To discover the root causes, systematic processes are proposed which provide suggestive performance anomaly solutions.

VII. Conclusion and Future Work

In this paper, we address a critical need in detecting peformance bottlenecks, relating them to known antipatterns and utilizing appropriate soultions. Our approach is based on suggestive profiling methods for different levels of abstractions. Common profiling include path-oriented profiling, thread behavior profiling, networking profiling, and system resource profiling. For each of these profiling methods, we include a suggestive profiling method, and we suggest alternatives for re-engineering the software system to achieve better performance. Request and workload profiles can also be generated through the suggestive profiling tool. This technique is used in the solution suggestion during refactoring phase of performance engieering, and is synchronized with software development cycles. The suggestive profiling tool and the framework utility tools have been implemented and demonstrated using RUBiS benchmark to evaluate performance bottlenecks.

There are limitations in matching some performance antipatterns with detectable features, and thus they cannot be detected directly. Most of these undetectable antipatterns are due to design decisions. Thus, an intimate knowledge of the designs can help in the detection and elimination of those performance antipatterns. If the design decisions can be systematically codified, then it will be possible to extend our framework to other performance antipatterns.

In the future, we plan to further analyze factors influencing antipatterns in different domains including high-performance computing, e-commerce or workflow data management, and extend the framework with appropriate tools. We also plan to make the framework Cloud-ready, so that general performance antipatterns in the computation of distributed systems can be categorized, detected, and resolved systematically.

Acknowledgment

This work is supported in part by the NSF Net-Centric and Cloud Software and Systems Industry/University Cooperative Research Center and award 1128344. The authors would also like to thank Dr. Shih-Kun Huang of National Chiao Tung University, Taiwan for his assistance with this project, and valuable comments and suggestions to improve the quality of the paper. We also acknowledge David Struble for his help in proofreading.

References

[1] R. Johnson, R. Helm, J. Vlissides, and E. Gamma, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1995.

[2] C. U. Smith and L. G. Williams, Performance solutions: a practical guide to creating responsive, scalable software. Addison-Wesley Reading, 2002, vol. 1.

[3] C. U. Smith and L. G. Williams, "Software performance antipatterns." in Workshop on Software and Performance, 2000, pp. 127–136.

[4] C. U. Smith and L. G. Williams, "New software performance antipatterns: More ways to shoot yourself in the foot," in Int. CMG Conference, 2002, pp. 667–674.

[5] C. U. Smith and L. G. Williams, "More new software performance antipatterns: Even more ways to shoot yourself in the foot," in Computer Measurement Group Conference, 2003, pp. 717–725.

[6] C. U. Smith, "Introduction to software performance engineering: Origins and outstanding problems," in Formal Methods for Performance Evaluation. Springer, 2007, pp. 395–428.

[7] T. Ball and J. R. Larus, "Efficient path profiling," in Proceedings of the 29th annual ACM/IEEE international symposium on Microarchitecture. IEEE Computer Society, 1996, pp. 46–57.

 ISBN: 978-1-61208-367-4

[8] J. Huang, “State constraints and pathwise decomposition of programs,” Software Engineering, IEEE Transactions on, vol. 16, no. 8, 1990, pp. 880–896.

[9] C.-K. Luk et al., “Pin: building customized program analysis tools with dynamic instrumentation,” in Acm Sigplan Notices, vol. 40, no. 6. ACM, 2005, pp. 190–200.

[10] Rubis web site. [Online]. Available: http://rubis.ow2.org/ [retrieved: August 2014]

[11] Apache http server project web site. [Online]. Available: http://httpd.apache.org/ [retrieved: August 2014]

[12] Jboss application server web site. [Online]. Available: http://jbossas.jboss.org/ [retrieved: August 2014]

[13] Mysql community server web site. [Online]. Available: http://dev.mysql.com/ [retrieved: August 2014]

[14] T. Parsons, “Automatic detection of performance design and deployment antipatterns in component based enterprise systems,” Ph.D. dissertation, Citeseer, 2007.

[15] C. Trubiani and A. Koziolek, “Detection and solution of software performance antipatterns in palladio architectural models.” in ICPE, 2011, pp. 19–30.

[16] F. Brosig, S. Kounev, and K. Krogmann, “Automated extraction of palladio component models from running enterprise java applications,” in Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, p. 10.

[17] V. Cortellessa, M. De Sanctis, A. Di Marco, and C. Trubiani, “Enabling performance antipatterns to arise from an adl-based software architecture,” in Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on. IEEE, 2012, pp. 310–314.

[18] V. Cortellessa, A. Di Marco, and C. Trubiani, “Performance antipatterns as logical predicates,” in Engineering of Complex Computer Systems (ICECCS), 2010 15th IEEE International Conference on. IEEE, 2010, pp. 146–156.

[19] A. Wert, J. Happe, and L. Happe, “Supporting swift reaction: Automatically uncovering performance problems by systematic experiments,” in Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, 2013, pp. 552–561.

[20] J. Xu, “Rule-based automatic software performance diagnosis and improvement,” Performance Evaluation, vol. 67, no. 8, 2010, pp. 585–611.

[21] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, “An analytical model for multi-tier internet services and its applications,” in ACM SIGMETRICS Performance Evaluation Review, vol. 33, no. 1. ACM, 2005, pp. 291–302.

[22] V. Cortellessa, A. Martens, R. Reussner, and C. Trubiani, “A process to effectively identify guilty performance antipatterns,” in Fundamental Approaches to Software Engineering. Springer, 2010, pp. 368–382.

[23] S. Iwata and K. Kono, “Clustering performance anomalies based on similarity in processing time changes,” IPSJ Online Transactions, vol. 5, no. 0, 2012, pp. 1–12.

[24] C. Stewart, K. Shen, A. Iyengar, and J. Yin, “Entomomodel: Understanding and avoiding performance anomaly manifestations,” in Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on. IEEE, 2010, pp. 3–13.

 ISBN: 978-1-61208-367-4

Performance Optimisation of Object-Relational Database Applications in Client-Server Environments

Zahra Davar*, Janusz R.Getta†, Handoko‡

School of Computer Science and Software Engineering, University of Wollongong
Email: *zd991@uowmail.edu.au, †jrg.@uow.edu.au
, ‡h629@uowmail.edu.au

***Abstract*—The optimisation of object-relational database applications implemented as a mixture of object-oriented and non-procedural code, requires accurate balancing of the data-processing load between the client side and the server side. When there are large amounts of procedural code and less efficient and overly simple algorithms, the majority of the data processing takes place on the client side. As a consequence it usually increases the amounts of data transmitted to the client side and also, the amounts of time needed to process data on the client side. This paper shows how object-relational database applications can be optimised through a better balancing of the data processing load between the client and the server sides. A collection of transformation rules is developed, which replaces the typical iterative structures of procedural code with the equivalent structures of non-procedural code. The software patterns proposed in the paper allow for the automatic optimisation of object-relational applications.**

Keywords–Object-Relational Application; Performance; Transformation Rule; Software Patterns.

I. INTRODUCTION

Object-relational mapping and the efficient implementation of object-relational applications, have recently received great deal of attention, especially in commercial environments [1]. Over the past decade, the performance of object-relational applications has become a serious challenge for programmers and database researchers [2].

An object-relational database application is a typical client/server application [3]. In relational systems, the majority of query processing is performed on the server side [4]. Object-relational mapping makes the relational database system available on the server side and visible to an application programmer as a collection of classes of objects on the client side. This means that, relational tables or stored procedures on the server side are wrapped into classes on the client side, so that objects and methods can be used on the client side as well. This is why object-relational database applications are typically implemented in the object-oriented programming language embedded with the simple non-procedural statements of the Object Query Language (OQL).

Programmers, access data on the client side through iterations over the classes of objects or over the results from the processing of OQL statements. Such an approach to the implementation of object-relational applications tends to reduce the amount of non-procedural code and to significantly simplify the code when accessing the object-oriented view of the database. For instance, a traversal of an associations between two classes of objects, is implemented as nested loops which iterate over the objects [5]. Programmers typically focus on the logic of an application rather than on how the data will be processed on the client side. This approach to the implementation of an object-relational application, is the main cause of two serious performance problems.

First, the iterations over the large classes of objects on the client side require transferring large amounts of data from the server side. Second, to process these data on the client side, a programmer uses the algorithms which are not as efficient as algorithms which can process the same data on the server side. For example, a traversal of an association on a client side is typically implemented as a join of two relational tables on a server side [5]. Then implementation of *join* operation on the server side with a hash-based or index-based algorithms is much more efficient than implementation of the same *join* operation on the client side with a nested loop algorithm.

Implementing efficient object-relational applications is a serious challenge. There are a number of ways to solve this problem. Recognising the control structures of an application so that it can be rebuilt with more non-procedural code is one solution. This means that, only the objects needed to satisfy the filtering conditions of the application are transferred from the server side to the client side. By using this approach, each relational application written by a programmer can be restructured so as to achieve the same results faster.

This paper, presents a set of transformation rules which can eliminate the iteration over a large number of objects and reduce the amounts of data transmitted over a network by changing the control structures of an application. By applying the rules, some of the procedural components are replaced with OQL statements. This results in faster and more efficient performance of the application. This paper also proposes software patterns which can be used by an application programmer. These patterns, allow the automatic optimisation of object-relational applications.

In the remainder of this paper, experimental results are presented in Section II to show the scale of the problem. Section III reviews the existing research on performance tuning of object-relational applications. The transformation rules are presented in Section IV. Software patterns for different styles of programs are presented in Section V. Section VI contains the conclusion and suggested future work.

II. CASE STUDIES/EXPERIMENTS

Experiments, were conducted using the TPC-H benchmark database which has 300 MB of relational data. The Lucid Lynx Ubuntu system running on 3.33GHz Intel(R), Core(TM)2, Duo CPU with 3.25GB RAM was used to run the applications. The examples were run in Java Persistence API (JPA) format

 ISBN: 978-1-61208-367-4

and in the NetBeans 7 environment. Various experiments were performed for different size databases and the run-time of the applications was measured, using NetBeans run-time clock. In all examples, the *Supplier* class, consists of 3000 objects and the *Lineitem* class varies between 400,000 objects to 1,800,000 objects.

In this section, two sets of experiments are presented to show the motivation of this research. In each set of experiments, the run-time of two different applications with the same output is measured and compared. In each set of experiment, the first application is the original inefficient version of the application and the second one is the improved version. The first set of experiments, presents *anti-join* traversal and the second, illustrates counting from aggregation applications.

A. Anti-Join Traversal

The following anti-join application, is used to identify suppliers whose products have never been ordered.

```
{Query query1 = em.createQuery
 ("SELECT  s FROM Supplier s", Supplier.class);
 List list1 = query1.getResultList();
 while(iterator1.hasNext())
{Query query2 = em.createQuery
 (" SELECT  l FROM Lineitem l
   WHERE l.L_SUPPKEY="
    +query1.getInt("s.S_SUPPKEY"));
List list2 = query2.getResultList();
Iterator iterator2= list2.iterator();
 boolean found = false;
 while (iterator2.hasNext())
 if (query1.getInt("s.S_SUPPKEY") ==
   query2.getInt("l.L_SUPPKEY")){
     found=true; } }
    if (!found) {
    System.out.println
    ("ITEM " + query1.getInt("s.S_SUPPKEY")
     + "not exist in LINEITEM ");
    found = false; } }
```

Figure 1. Application C.

To test the performance of Application C, the run-time of the application with different sizes of the class *Lineitem* is recorded.

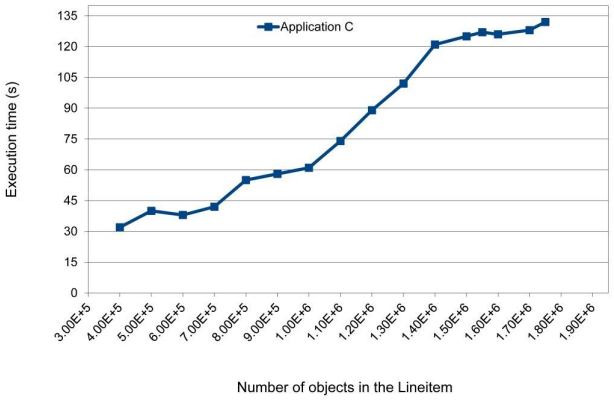


Figure 2. Execution Time for Application C.

Figure 2 shows the result of running the nested loop structure of the *anti-join* application with different size *Lineitem* classes. The run-time of Application C started from 30 seconds with 400,000 objects in the Lineitem class and increased to approximately 130 seconds for 1,700,000 objects.

Next, Application D is implemented using a *left outer join* clause. Using this, only the objects which satisfy the *anti-join* condition transfer to the client side. This *anti-join* application, has the same output as Application C but in the shorter run-time.

```
{Query query = (Query) em.createQuery
("SELECT s FROM Supplier s
  LEFT OUTER JOIN Lineitem l WHERE
    s.S_SUPPKEY=l.L_SUPPKEY", Supplier.class);
     List list1 = query.getResultList();
     Iterator iterator1= list1.iterator();}
```

Figure 3. Application D.

Figure 4 shows the run-time of Application D for different size *Lineitem* classes. The run-time was varied, between 2-4 seconds.

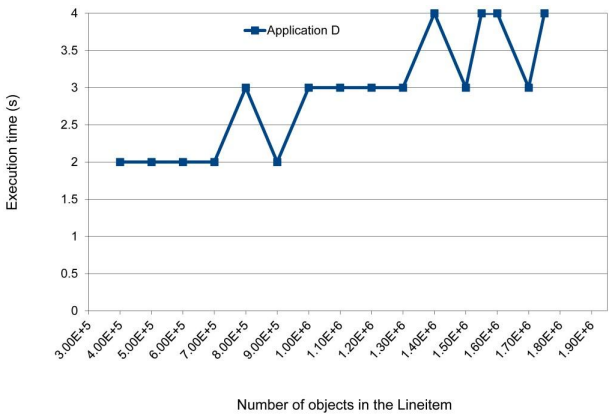


Figure 4. Execution Time for Application D.

Figure 2 and Figure 4 clearly show that Application D is much more efficient than Application C. The run-time for Application C, took 30 seconds with 400,000 objects in the *Lineitem* class and increased to approximately 2 minutes for 1,700,000 objects. Implementing the *anti-join* application with a *left-outer-join*, instead, caused a run-time of between 2 to 4 seconds. On average, Application D ran 26 times faster than Application C. Therefore, to implement an *anti-join* application for large database with complex objects, the implementation of Application D is more efficient than Application C.

B. Counting Objects

Two different applications were run for counting objects from a class. Both applications retrieve the same outputs. Application E is implemented with two *SELECT* statements, which iterates on the results of the first *SELECT* statement. This application finds the same objects in *Lineitem* class and counts them. Application E, is implemented by nested *SELECT* statements as follows:

```
{ Query query1 = em.createQuery
 ("SELECT  Distinct l_Suppkey FROM Lineitem l");
    List list1 = query1.getResultList();
    Iterator iterator1= list1.iterator();
    while(iterator1.hasNext())
     { Query query2 = em.createQuery
       ("SELECT  COUNT(*) FROM Lineitem l
         WHERE
           l_Suppkey= list1.l_Suppkey ");
       List list2 = query2.getResultList();
       Iterator iterator2= list2.iterator();}}
```

Figure 5. Application E.

Application E was run several times with different size of databases. The results of running Application E are set-out in Figure 6.

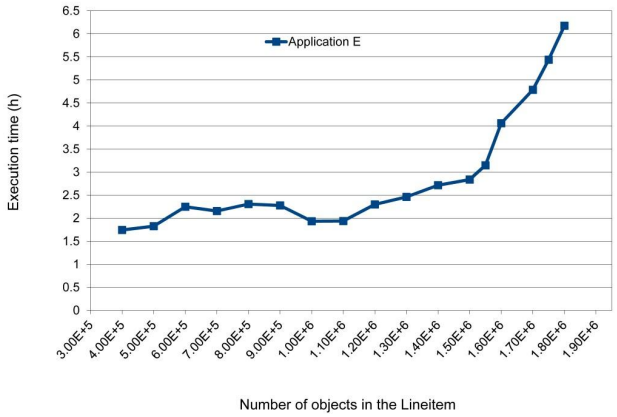


Figure 6. Execution Time for Application E.

Figure 6 shows that, the run-time of Application E took 1.5 hours for 400,000 objects and increased to around 6 hours for 1,800,000 objects.

Application F is implemented by reconfiguring Application E. Application F, used a *Group by* clause to group the results of counting the same objects and transfer them to the client side. This approach, eliminates the necessity for iteration over all of the objects in the class.

```
{Query query = (Query) em.createQuery
("SELECT l.l_Suppkey,
 COUNT(l.L_Suppkey) As total
   FROM Lineitem l
    GROUP BY l.l_Suppkey
     ORDER By total");
     Number countResult=
     (Number) query.getResult();}
```

Figure 7. Application F.

The results of running Application F with different sizes of *Lineitem* class is presented in Figure 8. The run-time of Application F varied between 3 and 5 seconds.

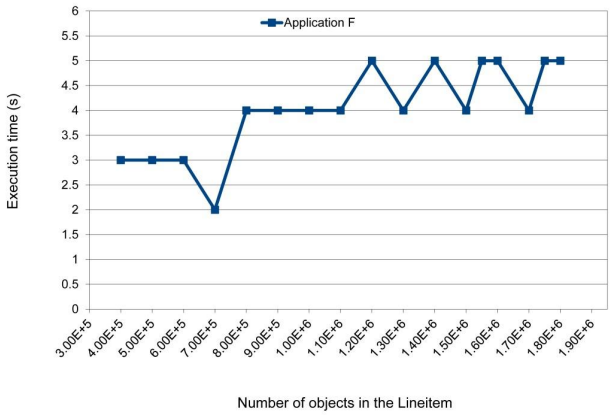


Figure 8. Execution Time for Application F.

A comparison of Figures 6 and 8 shows a very large performance difference between Application E and Application F. Application F was 55 times faster than Application E. The experimental results show that by reconfiguring object-relational applications so that fewer objects are transferred to the client side and more data-processing is done on the server side, better performance of the application is achieved.

III. Related Work

Agarwal [6] proposed the idea of using a client-side object cache in order to increase the performance of the application and suggested that the actual performance was greatly dependent on the degree to which the application can take advantage of data stored in the object cache. The problem in this method, however, is that the complexity of the query must be managed so that it can return instances of commonly used classes with minimum use of joins. In 2006, P. V Zyl et al. focused on comparing the performance of object databases and object-relational mapping tools. This research discussed object-relational mapping in open source applications [7]. This approach, however, only dealt with one framework and was not tried on the distributed or multi-user frameworks which are often used by developers. R. Kalantari et al. compared the performance of object and object-relational database systems. They suggested a number of factors which system developers must consider when selecting a database management system for persisting objects [8] but it was done based on basic query implementation which means that, it did not consider complex queries involving two or more objects. This also means that it is less than optimal for todays applications with complex queries. Rahayu et al. discussed the performance evaluation of object-relational transformation methodology. The aim of this research was to clarify the efficiency of the operations on relational tables based on certain object-relational transformation methodology [9]. The performance of object-relational transformation methodology was also compared with that of the conventional relational model. This work, however, did not involve the dynamic parts of the object orientation. Meng et al. proposed a some transformation rules for object-oriented database systems. The rules used in this research were designed to transform the structural part of an object-oriented database schema into an equivalent relational schema [10]. These rules provided a relational view of the object-oriented database schema for

relational users. This research is limited, however, to the structure of a relational front end for object-oriented database systems. The idea of translating queries from an SQL into an OQL in an automatic way, was suggested by Mostefaoui et al. Their method was based on graph representations [11]. A formal approach for translating object-oriented database queries into equivalent relational queries was proposed by Yu et al. who used the same method as Mostefaoui et al. [11][12]. These works, however, did not consider all the possible forms of SQL queries. In addition, the methods suggested were not general enough to be extended to other clauses and they could not address the performance problem of object-relational applications. Grust et al., developed the FERRY language which was designed as an intermediate language which acts as glue that permits a programming style in which developers access database tables using their programming language's own syntax and idiom [13]. In 2010, the same authors extended this approach by proposing the FERRY-based LINQ to SQL approach [14]. Both papers were based on compiling the first-order functional programs into SQL which is not an applicable approach in industry. Recently, Chen et al. proposed a framework which can detect and prioritise instances of object-relational mapping performance anti-patterns [15], and therefore, improve the systems response time. This is useful but this approach can detect performance bugs and leaves the debugging process for the developer. In our previous work, performance tuning of object-oriented applications in distributed frameworks was discussed. The structure of the proposed approach needs to be upgraded in order to be more efficient. Also, the approach in [16] is only applicable to distributed frameworks and the templates are not general enough to be applicable to complicate applications. It should also be noted that the idea presented in [16], was not sufficiently evaluated through experimental results.

IV. TRANSFORMATION RULES

The transformation rules presented in this paper convert the non-optimised version of the object-relational database applications into optimised ones in order to provide the necessary efficiency and high speed. The configuration of object-oriented application has been changed by replacing certain procedural parts of the code with non-procedural code. The transformation rules create equivalent applications, where less data is transferred from the server and more data-processing is done on the server side. The transformation rule is applied to the non-optimised version of the program which is an input component and the result is an optimised version of the program, which is an output component. By using more OQL code and changing the structure of the input component, the output component is implemented.

In this paper a filtering (selection) transformation rule, an association anti-traversal rule and an aggregation rule are presented. The rule for *Association Traversal* (presented in our previous work) is presented here, in order to make the JAVA pattern of this specific rule which is obtained in Section V meaningful. Except *Association Traversal* rule, other rules are designed based on the recent experimental results. In the following algorithms, a text *p r o c e s s i n g* means any block of *Java* code.

A. Selection/Filtering Transformation Rule

Each relational application, can include an iteration over one class of objects (selection) which filters the outputs. For this case, the configuration of the application is changed from a program with one *SELECT* statement and one *IF* clause (as shown in Algorithm 1), to a program with one *SELECT* statement and one *WHERE* clause (as shown in algorithm 2). Therefore, some procedural parts of the code, are replaced with non-procedural code. Figure 9 is the input component algorithm for the filtering rule.

Algorithm 1: Input component
Iteration over one class of objects

```
1 for each t in (SELECT * FROM Class ) do
2 |   if φ [t.t_1, t.t_2, ..., t.t_n] then
3 |   |   p r o c e s s i n g
4 |   end
5 end
```

Figure 9. Input component for Selection Rule.

Figure 10 presents the algorithm of the output component, after applying the above changes to the input component.

Algorithm 2: Output component
Filtering

```
1 for each t in (SELECT * FROM Class WHERE φ
  [t_1, t_2, ..., t_n] ) do
2 |   p r o c e s s i n g
3 end
```

Figure 10. Output Component for Selection Rule.

An example of $\varphi\ [t_1, t_2, ..., t_n]$ is: Class.Object$_i$=2. Non-relational conditions in the input component $\varphi\ [t.t1, t.t2, ..., t.tn]$, will convert to $\varphi\ [t_1, t_2, ..., t_n]$ in the output component. This means that, references to the objects $(t.t_1)$, are removed and the output component operates on the name of the properties (t_1). The structure of the entire expression and all contents, however, remain unchanged.

B. Association Anti-Traversal/Anti-Join Transformation Rule

A common input component algorithm, for the *anti-join* rule is introduced in Figure 11. The input component, includes a variable which is *False* by default. This variable will become true if the *SELECT* statement finds any object from class 2, which satisfies the same condition as the object in class 1. *Anti-join* applications, retrieve objects from the second class, which do not exist in the first class.

Algorithm 3: Intput component
Anti-Join by Variable

```
1 for each t in (SELECT * FROM Class 1) do
2 |   Found = False
3 |   for each s in (SELECT * FROM Class 2 WHERE
  |   Class 2.Member_j = Class1.t)do
4 |   |   Found = True
5 |   |   Exit
6 |   end
7 |   If not Found p r o c e s s i n g
8 end
```

Figure 11. Input Component for Association Anti-Traversal.

The algorithm of the output component for association anti-traversal is presented in Figure 12. Our experimental results show that the algorithm presented in Figure 12, can be used as an input component for any possible implementations of the *anti-join*'s input components.

Algorithm 4: Output component
Anti-Join by Left outer join

```
1 for each p in (SELECT * FROM Class1 Left Outer
  Join Class2 on Class2.Member_j = Class1.Member_i)
  do
2 |   if Class2.Member_j is Null then
3 |   |   p r o c e s s i n g
4 |   end
5 end
```

Figure 12. Output Component for Association Anti-Traversal.

In our approach, the output component is written by only one *SELECT* statement. The *left-outer-join* is used in the output component of this rule to select the objects with the same condition and makes it unnecessary to transfer them to the client side.

C. Aggregation

Based on our recent experimental results, the input component of the aggregation rule is designed as Algorithm 5. The rule, can support all different types of aggregation applications. The aggregation rule is based on finding similar objects in a class and then applying the aggregation function. For instance, for counting similar objects from a class of objects, F(x) can be a *COUNT(*)* in the input component and *COUNT(Member$_i$)* in the output component. Algorithm 5, is designed using nested *SELECT* statements. In this algorithm, F(x) is the function related to the specific aggregation type, which is used by the application developer. This function can be MIN, MAX, SUM, AVG, or COUNT.

Algorithm 5: Input component
Aggregation with nested loop

```
1 for each t in (SELECT a.Member_i FROM Class a) do
2 |   for each s in (SELECT F(x) FROM Class b
  |   WHERE b.Member_i=a.Member_i) do
3 |   |   p r o c e s s i n g
4 |   end
5 end
```

Figure 13. Input Component for Aggregation.

Algorithm 6 presented in Figure 14, used *Group by* clause to group the necessary objects and transfer them to the client side.

Algorithm 6: Output component
Aggregation with grouping objects

```
1 for each t in (SELECT Member_i, F(x) FROM Class
  Group by Member_i) do
2 |   x = getInt(Member_i)
3 |   y = resultset(F(x))
4 |   p r o c e s s i n g
5 end
```

Figure 14. Output Component for Aggregation.

By using the *Group by* clause, less objects will be transferred from the server side to the client side. This means that, less run-time is needed to run a application.

D. Iterations over two classes of objects/Association Traversal Rule

Algorithm 7, includes two nested *SELECT* statements which performs the *JOIN* operation. Algorithm 7 is the input component for association traversal applications.

Algorithm 7: Input component
Iterations over two classes of objects

```
1 for each t in (SELECT * FROM Class1 WHERE φ
  [t_1, ..., t_n]) do
2 |   for each s in (SELECT * FROM Class2 WHERE γ
  |   [s_1, s_2, ..., s_n] + γ' [< s_1, t_1 >, ..., < s_n, t_n >]) do
3 |   |   p r o c e s s i n g
4 |   end
5 end
```

Figure 15. Input Component for Association Traversal.

In the output component, two *SELECT* statements are merged into one *SELECT* statement with a *JOIN* clause. More non-procedural code is used to write the output component.

Algorithm 8: Output component
Join

1 **for** *each p in (SELECT * FROM Class1* ***Join*** *Class2 on*
2 γ' $[< t_1, s_1 >, < t_1, s_2 >, ..., < t_n, s_n >]$
3 ***Where***
4 φ $[t_1, ..., t_n]$ $\parallel$ γ $[s_1, s_2, ..., s_n]$ **do**
5 | p r o c e s s i n g
6 **end**

Figure 16. Output Component for Association Traversal.

An example of the function: γ' $[< s_1, t_1 >, < s_2, t_1 >, ..., < s_n, t_n >]$ in the input component is Class2.Member$_i$ = Class1.Member$_j$. They are the relational conditions of the application. The filtering conditions of Class2 are presented as γ $[s_1, s_2, ..., s_n]$. Concatenation in the output component can merge the non-relational conditions of both classes [16]. By using this rule, filtering conditions were applied to the objects on the server side and as a result, only necessary objects which satisfy the *JOIN* condition will transfer to the client side. This means that much data will remain on the server side. This leads to changing the balance of the data-processing between server and client and, as a result, enhances the performance of the application.

V. Software Patterns for Object-Relational Applications

An input component of any transformation rule, is a non-optimised version of an object-relational application. An object-relational application can be written in different ways, and these require a large number of rules to support and optimise them. To use the transformation rules, input components based on what most application programmers use, were needed. To solve this problem, number of software patterns in JAVA programming language have been suggested. The software patterns presented are based on new implementation of the rules and they are all standardised with the JAVA template. As long as the input component, which is the non-optimised version of the application is consistent with the following patterns, then the rules can be applied to the application and optimise it. Depending on the application, the name of the objects, the classes, the functions, the relational conditions and the non-relational conditions will change. Object-oriented programmers need to replace the statement inside $< >$, with the appropriate statement of their own code. The other parts of the pattern remain unchanged.

A. Selection/Filtering Template (SF.Temp)

The input component of the first rule in Section IV-A, must be be consistent with the following template:

```
{ Query query1 = em.createQuery(
  <Any SQL SELECT STATEMENT
      WHICH RETRIEVE OBJECTS>);
   List list1 = query1.getResultList();
   Iterator iterator1= list1.iterator();
    while(iterator1.hasNext()){
     {
      if <CONDITIONS> then
      <JAVA code>;
       }
}
```

Figure 17. Selection/Filtering Template (SF.Temp).

CONDITIONS can be any filtering conditions for the class. For instance: Class1.Object$_i$ =X

B. Association Traversal Template (AT.Temp)

If the input component matches this style, the output component of the rule for *iteration over two classes of objects*, which is presented in Section IV-D, is the optimised configuration of the application. In the algorithm, 'n' and 'n-1' are used to show the order of the tables. For instance, if there are two classes of objects, n must be considered as 2 and this means that the outer loop is analysed class 1, while the inner loop is analysed class2:

```
{ Query query'n-1' = em.createQuery
   <SQL SELECT statement from CLASS'n-1'>;
    <GET VARIABLE> ;
  <NON-RELATIONAL CONDITIONS of CLASS'n-1'>;
    List list1 = query'n-1'.getResultList();
    Iterator iterator1= list1.iterator();
    while(iterator1.hasNext()){
      <VARIABLE> =  query'n-1'.getInt(1);
      Query query'n' = em.createQuery
      <SQL SELECT statement from CLASS'n'>
      where <RELATIONAL CONDITIONS>;
       List list2 = query'n'.getResultList();
       Iterator iterator2= list2.iterator();
        while (iterator2.hasNext())
        {
     <NON-RELATIONAL CONDITIONS of CLASS'n'>;
         }
        <JAVA code>;
         }  }
```

Figure 18. Association Traversal Template (AT.Temp).

Assume that, 't' is an object variable which get the objects from the first class and 's' is another object variable which get the objects from the second class. Then: An example of 'NON-RELATIONAL CONDITIONS of CLASS1':
φ $[t.t_1, t.t_2, ..., t.t_n]$. Example: Class1.Object$_i$ =X.
An example of 'RELATIONAL CONDITIONS' :
γ' $[< s_1, t_1 >, ..., < s_n, t_n >]$. Example: Class2.Object$_j$=Class1.Object$_i$.
An example of 'NON-RELATIONAL CONDITIONS of CLASS2' :
γ $[s.s_1, s.s_2, ..., s.s_n]$. Example: Class2.Object$_j$=Y.

C. Anti-Join Template (AJ.Temp)

If the input component of the rule, matches this *anti-join* style, then it can be modified according to the rule, which is

presented in IV-B.

```
{ Query query1 = em.createQuery(
  <SQL SELECT statement from CLASS1>);
   GET VAR = FALSE;
   List list1 = query1.getResultList();
   Iterator iterator1= list1.iterator();
   while(iterator1.hasNext()){
     Query query2 = em.createQuery(
     <SQL SELECT statement from CLASS2>
      where
       <RELATIONAL CONDITIONS
          between CLASS1 and CLASS2>;
       List list2 = query2.getResultList();
       Iterator iterator2= list2.iterator();
       while (iterator2.hasNext())
        {
        if <ANTI-JOIN CONDITION>{
          VAR = True,
          Exit;
        } }
          if VAR=FALSE
             { <JAVA code>;
             }
        }
}
```

Figure 19. Anti-Join Template (AJ.Temp).

An example of <ANTI-JOIN CONDITION> is: list1.$Member_i$=list2.$Member_j$.

D. Aggregation Template (AG.Temp)

The general template to use the aggregation rule, is presented as below.

```
{ Query query1 = em.createQuery
 ("SELECT <a.Memberi> FROM <CLASS a>");
   List list1 = query1.getResultList();
   Iterator iterator1= list1.iterator();
   while(iterator1.hasNext())
    {
    Query query2 = em.createQuery
    ("SELECT <F(x)>
     FROM <CLASS b>
      WHERE
      <AGGREGATION CONTITION>);
        List list2 = query2.getResultList();
        Iterator iterator2= list2.iterator();
         while (iterator2.hasNext())
        {
       if <AGGREGATION CONTITION> then
         }
         <JAVA code>;
          }  }
```

Figure 20. Aggregation Template (AG.Temp).

To use the output component presented in IV-C, the application must match the following style. F(x) can be any type of the aggregation: MIN , MAX , SUM , AVG or COUNT.

E. n Associations Template

Assume that F: Filtering, C: Condition, J: Java Code, JC: Join Conditions, V: Variable, A: Array, AGC: Aggregation Conditions and OV is an object variable which can keep the results from one template and passes it to the other template. The object variable, takes the results from each template and pass it to the next template. At the end of each template, the object variable is updated to the new object variable which includes new results from the current template and this object template is ready to use in the next template. Therefore, generally all the templates can be presented as:
$\mathrm{T}_F < F, C, OV, J >$
$\mathrm{T}_{TA} < F_1, F_2, C, JC, OV, J >$
$\mathrm{T}_{AJ} < F_1, F_2, C, OV, J >$
$\mathrm{T}_{AG} < A, F, AGC, OV, J >$

For *n* association, the template will be a mixture of the above templates. To mix the templates, '*J*' must be replaced with the desire template. Also, *OV* from the first template must pass to the next template. The inner most class is considered as class 2. For instance, let us assume that there are 3 classes: Student Name(Class1), Course(Class2), Marks(Class3) and the programmer would like to find all the student names which start with A and then find who does not take Maths and then find who gets a mark above 50 in other courses. In this case, there is a filtering at the beginning for class1, then an antijoin of class1 and class2 and at the end association of traversal between class2 and class3. By using the above short templates, the following template was written for this example:
$\mathrm{T}_F < F, C, OV, < T_{AJ} >> \text{——->} \mathrm{T}_{F,AJ} < F, C, < OV, F_1, F_2, C, OV, J >> \text{——->} \mathrm{T}_{F,AJ,TA} < F, C, < OV, F_1, F_2, C, JC, < OV, F_2, F_3, C, JC, OV, J >>>$

Now by replacing each actual template instead of the name of the template, the final template will design. To make the above example more applicable, the actual templates are replaced in the last achived template ($\mathrm{T}_{F,AJ,TA} < F, C, < OV, F_1, F_2, C, JC, < OV, F_2, F_3, C, JC, OV, J >>>$).

```
{\\Filtering template\\
Get OV;
Query query1 = em.createQuery(
<Any SQL SELECT statement
which retrieve objects>);
List list1 = query1.getResultList();
Iterator iterator1= list1.iterator();
while(iterator1.hasNext())
{if <CONDITIONS> then
 {\\Anti join template\\
  Get OV;
  Query query2 = em.createQuery(
  <SQL SELECT statement from CLASS1>);
  GET VAR = FALSE ;
  List list1 = query2.getResultList();
  Iterator iterator2= list1.iterator();
  while(iterator2.hasNext())
  {Query query3 = em.createQuery(
   <SQL SELECT statement from CLASS3>
    where <RELATIONAL CONDITIONS
     of CLASS1 and CLASS2>;
    List list2 = query3.getResultList();
    Iterator iterator3= list2.iterator();
    while (iterator3.hasNext())
    {VAR = True;
    Exit;}}
```

```
if VAR=FALSE {\\Join template\\
 Get OV;
 Query query4 = em.createQuery(
 <SQL SELECT statement from CLASS1>);
 <GET VARIABLE> ;
 <NON-RELATIONAL CONDITIONS of CLASS1>;
 List list4 = query4.getResultList();
 Iterator iterator4= list1.iterator();
 while(iterator4.hasNext())
 {<VARIABLE> =  rset4.getInt(4);
 if <CONDITIONS> then{
 Query query5 = em.createQuery(
 <SQL SELECT statement from CLASS2>
 where
  <RELATIONAL CONDITIONS
    of CLASS1 and CLASS2>);
 List list5 = query5.getResultList();
 Iterator iterator5= list5.iterator();
 while (iterator5.hasNext()){
 if <NON-RELATIONAL CONDITIONS
    of CLASS2> then
 <JAVA code>; } } } } } } }
```

Figure 21. n Associations Template.

After obtaining the final design of the input component template, the rules can be applied. To do this, the developers must first find out which styles their application consist of. Then they can use the *n* Associations Template to build their application step by step. Then the rules can be applied to the application and the result is the optimised version of the application. By replacing the short form with the actual template, the final template is achieved as above.

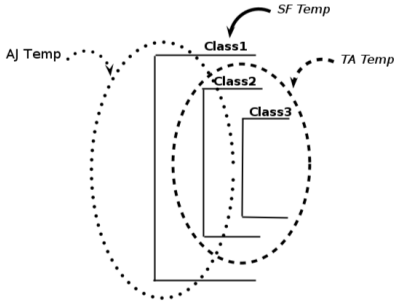


Figure 22. Use templates to prepare the input component

Figure 22, shows how theoretically the templates can be applied to the above example. The templates, however, were applied on the real applications, but the presentation of these, is beyond the scope of this paper.

VI. Conclusion and Future Work

This paper attempted to solve the problem of implementing efficient object-relational applications in client-server frameworks. Certain transformation rules, which can shift more data-processing to the server side, have been proposed. Using this approach, decreases the amount of data transfer from the client side to the server side. Therefore, only the essential objects will be transferred from the server side to the client side. Software patterns of the rules are also presented to make the rules more applicable. The correctness of the rules did not fit in the scale of this paper. Future work will introduce a support tool, which can apply the patterns automatically to the applications.

VII. Acknowledgments

The authors wish to gratefully acknowledge the help of Dr. Madeleine Strong Cincotta in the final language editing of this paper.

References

[1] J. Orsag, "Object relational mapping", D. thesis, Comenius University, Bratislava, Slovakia, 2006.

[2] J. Duhl and C. Damon, "A performance comparison of object and relational databases using the Sun Benchmark", In Proc. ACM Conference on object-oriented programming systems, languages and applications (OOPSLA '88), New York:Norman Meyrowitz, 1998, pp. 153-163.

[3] S. Son, I. Yoon, and C. Kim, "A Component-Based Client/Server Application Development Environment using Java", In Proc. IEEE Computer Society Conference on the Technology of Object-Oriented Languages and Systems (TOOLS '98), Washington, 1998, pp. 168.

[4] M. J. Franklin, B. T. Jnsson, and D. Kossmann, "Performance tradeoffs for client-server query processing", In Proc. of the ACM SIGMOD international conference on Management of data (SIGMOD '96), 1996, Jennifer Widom (Ed.), New York, pp. 149-160.

[5] R. Ramakrishnan and J. Database Management Systems. New York: McGraw-Hill, 2002.

[6] S. Agarwal, "Architecting Object Applications for High Performance with Relational Databases", In OOPSLA Workshop on Object Database Behavior, Benchmarks, and Performance, Persistence Software, Inc, 1995.

[7] P. van Zyl, D.G. Kourie, and A, "Comparing the performance of object databases and ORM tools.Boake", In Proceedings of the annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries (SAICSIT '06), 2006, Judith Bishop and Derrick Kourie, pp. 1-11.

[8] R. Kalantari and C. H. Bryant, "Comparing the performance of object and object relational database systems on objects of varying complexity", In Proceedings of the 27th British national conference on Data Security and Security Data (BNCOD'10), 2010, Lachlan M. MacKinnon (Ed.), Springer-Verlag, Berlin, Heidelberg, pp. 72-83.

[9] J. Wenny Rahayu, E. Chang, T. Dillon, and D. Taniar, "Performance evaluation of the object-relational transformation methodology," Data Knowl. Eng, vol. 38, no. 3, pp. 265-300, 2001.

[10] W. Meng, C. T. Yu, W. Kim, G. Wang, T. Pham, and S. Dao, "Construction of a Relational Front-end for Object-Oriented Database Systems", In Proc. IEEE Computer Society Conference, 1993, pp. 476-483.

[11] A. Mostefaoui, and J. Kouloumdjian, "Translating Relational Queries to Object-Oriented Queries According to ODMG-93.", In: ADBIS, Springer, 1998, pp. 328-338.

[12] C. Yu, Y. Zhang, W. Meng, W. Kim, G. Wang, T. Pham, and S. Dao, "Translation of Object-Oriented Queries to Relational Queries", In: Proc. of the 11th Int. Conf. on Data Engineering, 1995, pp. 90-97.

[13] T. Grust, M. Mayr, J. Rittinger, and T. Schreiber "FERRY: database-supported program execution", In Proceedings of the ACM SIGMOD International Conference on Management of data (SIGMOD '09), Carsten Binnig and Benoit Dageville, ACM, New York, 2009, pp. 1063-1066.

[14] T. Schreiber, S. Bonetti, T. Grust, M. Mayr, and J. Rittinger, "Thirteen new players in the team: a FERRY-based LINQ to SQL provider", Journal of VLDB, vol. 3, no.2, pp. 1549-1552, 2010.

[15] T. Chen, W. Shang, Z. M. Jiang, A. E. Hassan, M. Nasser, and P. Flora, "Detecting performance anti-patterns for applications developed using object-relational mapping", In Proc. the 36th International Conference on Software Engineering (ICSE 2014), ACM, New York, USA, 2014, pp. 1001-1012.

[16] Zahra Davar, Janusz R Getta, "Performance Tuning of Object-Oriented Applications in Distributed Information Systems" Presented in 16th International Conference on Enterprise Information Systems (ICEIS 2014), Lisbon, Portugal, 2014.

 ISBN: 978-1-61208-367-4

Web Accessibility on Thai Higher Education Websites

Rattanavalee Maisak
Edith Cowan University
School of Computer and Security Science
Perth, Western Australia
r.maisak@our.ecu.edu.au

Justin Brown
Edith Cowan University
School of Computer and Security Science
Perth, Western Australia
j.brown@ecu.edu.au

***Abstract*—This paper examines web accessibility compliance in a sample of universities in Thailand. The Thai government has made a commitment to higher education and e-learning and has also signed on to the Convention on the Rights of Persons with Disabilities (CRPD). This paper shows that the web accessibility does not appear to be adopted by the universities examined in this study, with minimal compliance to the W3C Web Content Accessibility Guidelines 2.0. In particular, the Perceivable and Operable aspects of the guidelines seemed problematic in terms of the numbers of reported accessibility errors.**

Keywords-web accessibility, higher education, WCAG, evaluation.

I. INTRODUCTION

In the context of a web connected world, it falls to governments and organizations to focus on how they develop online information that is accessible to all, including people with disabilities. The World Health Organization (WHO) estimates that there are over 600 million people with disabilities living across the world moreover, some of those people experience barriers in accessing information and communication technologies [1]. As a result, countries such as Australia, the United States of America and Canada have developed and promoted web accessibility in their policies in order to minimize the barriers which prevent disabled people from participating in those societies. This paper examines accessibility in the Thai context, as the Thai government has made a formal commitment to the development of e-learning in Thailand as well as that of web accessibility.

The Thai government has been a signatory to the Convention on the Rights of Persons with Disabilities (CRPD) since 2007 [2]. In terms of the CRPD, the Thai government devotes significant resources to ensuring that people with disabilities in Thailand have equal rights with others. The Rehabilitation of Disabled Persons Act 1991 and the National Education Act 1999 were passed in order to improve learning opportunities of Thai students with disabilities [3][4]. However, research indicates that there is still a lack of educational facilities for students with disabilities in Thailand resulting from insufficient law enforcement and negative attitude of stakeholders [5]. Consequently, the needs of students with disabilities are not adequately supported through educational options. It is the belief of these authors, and of the wider literature that students with disabilities should be treated as equals, especially in the context of access to education suitable to their needs. As higher learning predominately takes place within a nation's university sector, this research aims to investigate the accessibility of Thai higher education websites. The rationale is that the level of accessibility of these university websites will in some small way reflect Thailand's adherence to its own legal requirements for equitable access to higher education.

II. RESEARCH QUESTIONS

This paper examines the research questions of 'what level of web accessibility is apparent in Thai universities against WCAG 2.0 guidelines' and 'against which aspects of the WCAG are the most issues seen'? The paper will address these questions by examining a number of Thai university websites, looking at both the main university webpages and publically visible pages containing e-learning content.

III. LITERATURE REVIEW

A. Accessibility Guidelines

Web accessibility is not a new concept and one can trace its origins back to the mid twentieth-century. The Americans with Disabilities Act (ADA) approved by the U.S. Congress in 1990 cemented protections against discrimination to Americans with disabilities. It guarantees equal opportunity for individuals with disabilities in terms of public transportation (in Title II), public accommodations (in Title III) and telecommunications relay services (in Title IV) [6]. The potential protections of the Rehabilitation Act, ADA and the popularization of the Internet support the need to make the web accessible [7].

The World Wide Web Consortium (W3C) created the Web Accessibility Initiative (WAI) on 7 April 1997 in order to develop protocols and guidelines that ensure the web for all [8]. The Web Content Accessibility Guidelines (WCAG) were created in order to promote the accessibility of websites. The WCAG 1.0 guidelines were released on 5 May 1999 and were replaced with WCAG 2.0 in 2008. The aim of WCAG 2.0 is to increase the accessibility of websites for people with disabilities according to the four POUR principles [9].

"Perceivable" means that web content and user interface modules must be offered to users in multiple formats in order

 ISBN: 978-1-61208-367-4

to meet users' perceptions. “Operable” means that the user interface and navigation components should be designed in a way that they work properly, especially in terms of assistive technologies. “Understandable” is about making websites understandable in terms of language levels, correcting errors and predictability. “Robust” refers to the capacity of websites to be interpreted by a variety of user agents.

WCAG 2.0 is divided into 12 guidelines, which are classified according to three conformance levels (A, AA and AAA) in order to respond to the different needs of people with disabilities. Moreover, WCAG 2.0 concerns problems of interference on the use of the page by unsupported technologies through four critical failures:

1. Checkpoint 1.4.2 Audio control - the control over audio is available to pause or stop.
2. Checkpoint 2.1.2 Keyboard trap - keyboard focus can be moved throughout webpage.
3. Checkpoint 2.2.2 Pause, stop or hide – providing the control over moving, blinking, scrolling, or auto-updating information which display more than five seconds.
4. Checkpoint 2.3.1 Three flashes or below threshold – content must not flash more than three times in one second.

WCAG claims that if a webpage does not meet those requirements, then people with disabilities will not be able to utilize the content of the given web page. There is a body of literature which argues that WCAG 2.0 provides guidelines only, and that they are not definitive when it comes to deciding if a webpage is accessible or not. However, this paper will be using WCAG 2.0 as the standard against which webpages will be evaluated.

B. Web Accessibility Evaluation

Automated tools are powerful evaluation tools for locating inaccessible elements in a website, however, relying on one tool cannot guarantee the accessibility of whole website [10]. The W3C provides a list of over 100 checkers in term of “Complete List Web Accessibility Evaluation Tools”, with these tools ranging from page-at-a-time checkers through to whole of site conformance evaluators [11]. Generally, automated tools are based around web accessibility standards and can typically be locally installed on a desktop environment, as a cloud service or as a local browser extension. One of the advantages of cloud based tools is that they do not platform specific as are most browser based plug-ins.

Manual testing is an alternative method, though in most cases should be considered a complimentary method to automated assessment. The evaluation of websites for accessibility requires human inspection moreover, W3C also recommends the combination of expertise and users in evaluation processes [12]. Automated tools are also troublesome in terms of presenting false positives and false negatives, issues that can usually be overcome by human evaluation. Obviously, the trade-of is that an automated assessment can cover a lot of pages or an entire site in a short period of time, whereas human evaluations can typically address only a small number of the total pages in a site.

C. Related Work

The evaluation of web accessibility for top international university websites reported that the websites of universities across different countries and regions had accessibility issues [13]. The study utilized a multi method approach using four automatic tools and manual tests, with the websites being selected from Times Higher Education Ranking. By looking at the average accessibility errors, universities in Asia were the most inaccessible websites followed by North America, Europe and Oceania (Australia and New Zealand). Moreover, the results when examining university policy indicated that less than half of those policies provide specific technical actions for resolving accessible websites issues. This implies that university websites may not be reliable resources to find accessibility solutions.

A study comparing the accessibility of one hundred universities’ website in The United States of America indicated that the university webpages failed to meet basic WCAG 1.0 guidelines, especially priority 2; moreover, the university homepages had the highest number of errors [14]. The authors suggest that universities should ensure compliance with web accessibility standards such Section 508 and WCAG guidelines in order to support services and facilities available to students with disabilities. However, the study used the superseded WCAG 1.0 guidelines and an automated tool called “Test Accesibilidad Web (TAW)”.

A study of the accessibility of Spanish university websites demonstrated that there was low level of accessibility conformance on Spanish university websites with 95.50% of webpages failing to meet the UNE 139803 [15]. The UNE 139808 is based on WCAG 1.0 and is the Spanish government’s policy document regarding accessible web content. Moreover, the study showed that over 60% of webpages failed HTML and CSS Validations. Again, this study use automatic tools, such as TAW, the W3C Validation Service and the W3C CSS Validation Service.

The investigation of accessibility in 20 Finnish higher education‘s institutes websites revealed that the tested websites had low inaccessibility levels in priority 1 (14 websites) and priority 2 (12 websites), followed by the high inaccessibility levels in priority 2 (8 websites) and the full accessibility level in priority 1 (6 websites) [16]. The study used TAW with the recommendations of Finnish Quality Criteria for web Services which is based on WCAG 1.0 and is published by the Finnish government. The authors suggest that those websites should be modified in order to achieve the full accessibility level (as defined in their study).

Finally, a study evaluating web accessibility and usability at Thailand Cyber University (TCU) for totally blind users stated that none of the selected webpages met a minimum requirement of WCAG 2.0 in automated testing [17]. The author claims that TCU which is the biggest e-learning provider in Thailand and that the entire website has endemic accessibility problems.

 ISBN: 978-1-61208-367-4

IV. RESEARCH DESIGN

A. Scope of Investigation

From the larger study on which this paper is based, nine higher education websites in Thailand were selected from the top five ranked Thai universities [18], two open universities, one special college for students with disabilities and one online institution. We chose this list as it offered a variety of institutions and delivery modes, from online only through to mixed mode and disability specific. Seven representative webpages were tested from within each of the university web sites, including the university homepage, library, webmail login page, contact us, e-learning portal homepage, e-learning forums and publically available e-learning content. There is a significant body of literature regarding web accessibility auditing methodology, most of which indicate that common pages (that most web sites would have) should be evaluated as a priority. The Website Accessibility Conformance Evaluation Methodology (WCAG-EM) 1.0 also indicates that accessibility evaluation should include the common webpages of the target website such as homepage, login page and contacts page [19]. Moreover, homepage and level one (all links visible from the homepage) of a website are appropriate for accessibility auditing [20][21]. As this paper examined university web pages, an effort was made to audit pages from the university websites as well as any available e-learning content, which are typically housed in systems different to those containing the main university website. The universities examined in this study represented a number of different types, including;

- Common: primarily an on-campus teaching mode with some support for online delivery
- Open: primarily an online institution, with some required on-campus delivery, such as exams and tests.
- Online: a purely online teaching environment.
- Special: mixed mode of delivery aiming at supporting students with a variety of disabilities

The breakdown of the four groups in this study in terms of web pages (N = 189) examined by automated assessment is Common (105 webpages), Open (42 webpages), Online (21 webpages) and Special (21 webpages).

B. Method

A number of researchers suggest the advantages of combining automated and manual testing techniques in order to ascertain the level of accessibility of websites [12][22]. Automated testing is driven by those systems that can scan a web page or an entire site and report on the errors that can be tested without human intervention, such as issues with alt text, color contrast and markup validation. Manual testing sees 'expert' human evaluators examine a smaller subset of pages, looking at visual and coding elements to see where violations against WCAG 2.0 exist, as well as where actual usability issues may be apparent. The webpages were evaluated by automated and manual testing based on WCAG 2.0 guideline at level A and AA.

SortSite [28] was used to audit all pages at level one of each of the websites (i.e., all pages linked from the homepage), whilst WAVE was used on each of the seven pages being examined (as WAVE is page-at-a-time tool) manually. Manual evaluations were conducted for the same seven pages, and for each of the seven pages we counted the number of failures identified based on WCAG requirements, with the results being categorized by POUR principles. Table I shows the breakdown of WCAG 2.0 in terms of checkpoints, for A and AA only. As Table I illustrates, the Perceivable principle contains the most checkpoints (and points of potential failure), with nine at level A and five at level AA, through to Robust with only two checkpoints at level A.

TABLE I. POUR PRINCIPLES CHECKPOINTS ACROSS LEVELS A-AA

Principle	Level A	Level AA	% of total
Perceivable	9	5	36.84
Operable	9	3	31.58
Understandable	5	5	26.32
Robust	2	0	5.26
Total	25	13	100%

The scoring values are '0' and '1', with a '0' score meaning that no violation of a checkpoint was identified using either the automated tools or via manual assessment. A score of '1' did indicate that the page in question produced a violation of a given WCAG 2.0 checkpoint. Figure 1 shows an example of the data collection against the POUR principles using the various testing methods used in this research, being automated multi-page, automated page at a time and manual assessment.

WCAG 2.0 Principles and Guidelines	Level	Automatic		Comment	Manual	Comment
		WAVE Score	SortSite Score		Score	
1. Principle: Perceivable						
1.1. Text Alternatives						
1.1.1. Non-text Content -	A	1	1		1	Missing all ALT text.
1.2. Time-based Media: Provide alternatives for time-based media						
1.2.1. Audio-only and Video-only (Prerecorded)	A					
1.2.2. Captions (Prerecorded)	A					
1.2.3. Audio Description or Media Alternative (Prerecorded)	A				1	No CC on Youtube
1.2.4. Captions (Live)	AA					
1.2.5. Audio Description (Prerecorded)	AA					
1.3. Adaptable						
1.3.1. Info and Relationship	A	1	1		1	Incorret HTML coding the text represents as a list.

Figure 1. Example of data collection

V. RESULTS AND DISCUSSION

The collected data was tested for normal distribution. The Shapiro-Wilk test was used and found to be non-normally distributed with $p < .05$ [30]. Therefore, the analysis used nonparametric tests for comparing the differences in the

mean scores. The focus of the analysis was on the errors found overall as well as the distribution of errors across the POUR principles.

TABLE II. MEAN OF POUR VIOLATIONS

	P	O	U	R
Chi-Square	12.888	10.946	4.487	4.144
df	3	3	3	3
Asymp. Sig.	**.005**	**.012**	.213	.246

a. Kruskal Wallis Test
b. Grouping Variable: course mode

Table II demonstrates that there were statistically significant differences in mean scores of the webpages regarding POUR violations at Perceivable, X^2 (3, N = 189) = 12.89, p = .01, Operable, X^2 (3, N = 189) = 10.95, p = .01 however, there were not statistically significant differences in mean scores of the webpages regrading the course mode category at Understandable, X^2 (3, N = 189) = 4.49, p = .21 and Robust, X^2 (3, N = 189) = 4.14, p = .25. It can be interpreted that there are differences in the mean scores of the webpages (in terms of errors) regarding POUR violations at Perceivable and Operable principles, but not Understandable and Robust principles, which could be caused by the lower number of checkpoints and thus lower number of potential error points available within these principles. A Post hoc test was conducted to determine which university groups are different (Table III).

TABLE III. VIOLATIONS PER PAGE AGAINST UNIVERSITY TYPE

GROUPS		P	O	U	R
Common	Mean	**2.74**	2.18	1.86	.75
	N	93	93	93	93
	SD	1.950	1.628	1.486	.816
Open	Mean	2.15	1.94	1.39	.61
	N	33	33	33	33
	SD	1.839	1.478	1.116	.788
Online	Mean	2.00	**2.50**	1.64	.57
	N	14	14	14	14
	SD	1.301	1.557	1.550	.756
Special	Mean	**1.22**	**1.06**	1.17	.33
	N	18	18	18	18
	SD	.943	1.211	1.200	.485

By looking at the average errors in the POUR principle per page, most errors were found at the Perceivable followed by Operable, Understandable and then Robust principles. The Perceivable and Operable outnumber Robust violations, with almost double the number of errors. For example, the Common group had the average error at Perceivable (2.74 errors per page) and Operable (2.18) compared to Robust (0.75). This dataset indicates that most Thai institution websites have common accessibility problems related to providing information in multiple formats and lack awareness of control over the web interface (see Figures 2 and 3). By looking at the different course modes, the results indicate that the Special group performs the best in terms of web accessibility with the lowest numbers of errors at all POUR principles with average 1.22, 1.06, 1.17 and 0.33 respectively (see Table III). This may be because the special institutions are strongly committed to providing accessible resources and educational services for students with disabilities and is perhaps not surprising that those webpages contain content which is fit for purpose.

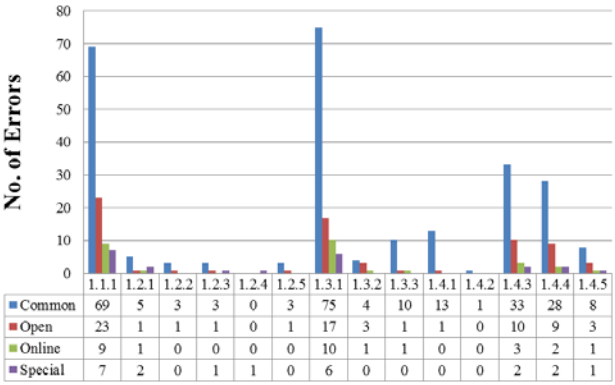
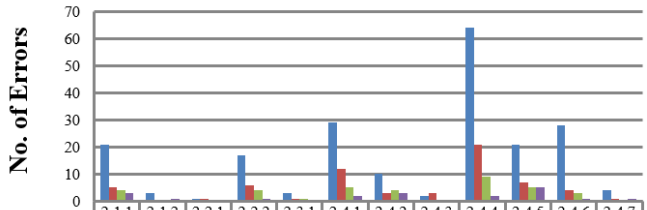


Figure 2. Breakdown of violations within Perceivable

By looking at the Perceivable principle, the highest total numbers of errors were found at checkpoints 1.1.1-Non text Content and the checkpoint 1.3.1-Info and Relationships with 108 errors (see Figure 2). Furthermore, the highest number of errors were found at checkpoint 1.1.1-Non text Content with 69 errors (Common), 23 errors (Open), 9 errors (Online) and 7 errors (Special) and the checkpoint 1.3.1-Info and Relationships with 75 errors (Common), 17 errors (Open), 10 errors (Online) and 6 errors (Special) respectively. This implies that most Thai institution websites have serious problems related to alternatives for non-text content and web structure and relationships (such as inconsistent use of heading styles to denote page structure).

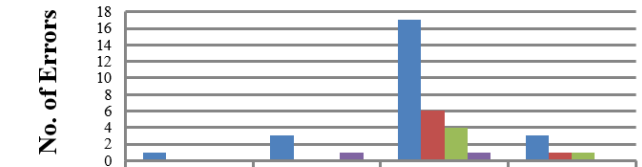
 ISBN: 978-1-61208-367-4



	2.1.1	2.1.2	2.2.1	2.2.2	2.3.1	2.4.1	2.4.2	2.4.3	2.4.4	2.4.5	2.4.6	2.4.7
Common	21	3	1	17	3	29	10	2	64	21	28	4
Open	5	0	1	6	1	12	3	3	21	7	4	1
Online	4	0	0	4	1	5	4	0	9	5	3	0
Special	3	1	0	1	0	2	3	0	2	5	1	1

Figure 3. Breakdown of violations within Operable

By looking at the Operable principle, the highest total numbers of errors were found at the checkpoint 2.4.4-Link purpose with 96 errors (see Figure 3). The breakdown of errors at checkpoint 2.4.4-Link purpose were 64 errors (Common), 21 errors (Open) and 5 errors (Online and Special). The dataset indicates that the websites would have critical problems in terms of descriptive links, with the most prevalent issue being the 'read more' link issues, which users of assistive technologies find to be singly uninformative.

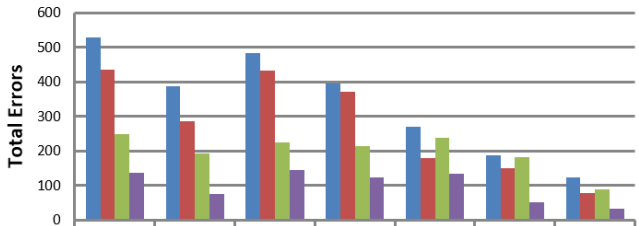


	1.4.2	2.1.2	2.2.2	2.3.1
Common	1	3	17	3
Open	0	0	6	1
Online	0	0	4	1
Special	0	1	1	0

Figure 4. Breakdown of critical failures

By looking at the critical failures, most errors were found at the checkpoint 2.2.2 followed by 2.3.1, 2.1.2 and 1.4.2 for all groups. The most errors were found at the checkpoint 2.2.2 Pause, Stop, Hide with 17 errors (Common), 6 errors (Open), 4 errors (Online) and 1 errors (Special) (see Figure 4). The dataset indicates that Thai institutions have problems involving the control over moving, blinking, scrolling, or auto-updating information which displays for more than five seconds – especially in terms of slideshows and carousels found on website homepages.

Figure 5 indicates the total number of errors found across the nine university sites for the seven pages tested in each of those sites. The results show that the university homepages had the most number of accessibility errors, not an uncommon finding in the literature [23][24] followed closely by the library pages. This is due in part to the number of links, content and multimedia items that both of these pages tended to contain, with contact us pages being problematic due to poor form design (lack of labels) and that the correct page language was not indicated (having English instead of Thai). Whilst the latter issue is not something every user would notice, lack of correct language identification for a page is an automatic fail of the Understandable principle, level A.



	University Homepage	E-learning Homepage	Library	Contact Us Page	Webmail Login Page	E-learning Forum	E-learning Content Page
P	529	387	483	396	270	189	125
O	435	287	433	371	179	151	78
U	248	193	225	215	239	182	89
R	136	77	144	124	134	52	32

Figure 5. Total violations across tested pages

The e-learning pages that were tested tended to fair better than the main university pages due to their general lack of multimedia content, with most of the content being text based materials coming out of e-learning tools such as Moodle [29]. Content management systems such as Moodle also address WCAG 2.0 guidelines to varying degrees [25], which is likely to have also contributed to a slightly higher level of accessibility for these pages as opposed to the main university pages.

VI. CONCLUSIONS

Whilst this paper represents the analysis of part of a larger research project, it does show that in terms of the Thai university system, there is still much work to be done in the web accessibility space. The results indicate that these university websites have accessibility problems even though the Thai government has signed the Convention on the Rights of Persons with Disabilities (CRPD). In particular, special institutions were created to provide an accessible online learning experience for students with a variety of disabilities, though in this study it seems even the web pages in that site were far from accessible (though better than the other non-specialist sites).

The limitation of this research is that the number of tested webpages is relatively small because there are only two open universities and one online institution in Thailand, which were examined in this study along with the top five universities. Moreover, the limitation of scoring method is that is essentially a binary one or zero, picking up the presence of an error but not the specifics of the error (which will be detailed in the larger study). However, we feel that

 ISBN: 978-1-61208-367-4

the data presented here indicates that Thai universities are not offering an accessible online learning experience to people with disabilities. Whilst the possible causes of the current situation are beyond the scope of this paper, it seems likely they are the same as other institutions across the globe, including lack of awareness, lack of policy and lack of WCAG 2.0 implementation, testing and knowledge [26][27]. We recommend that the Thai government strengthen its policy and requirements around accessibility of online technologies, and that this policy is clearly communicated to stakeholders in the government and education arenas.

VII. REFERENCES

[1] World Bank, World report on disability, in Main report 2011.

[2] The United nations, Convention on the Rights of Persons with Disabilities n.d.

[3] S. L. Carter, "The Development of Special Education Services in Thailand," International Journal of Special Education, vol. 21, pp.32-36, 2006.

[4] Commission, O.o.t.N.E., National Education Act of BE 2542 (1999). Bangkok: Seven Printing Group, 1999.

[5] T. Cheausuwantavee and C. Cheausuwantavee, "Rights, Equality, Educational Provisions and Facilities for Students with Disabilities in Thailand: Legal and Practical Perspectives over the Past Decade," Disability, CBR & Inclusive Development, vol. 23, pp.70-91, 2012.

[6] The U.S. Department of Justice, A Guide to Disability Rights Laws, 2009.

[7] E. Ellcessor, "Bridging Disability Divides: A critical history of web content accessibility through 2001," Information, Communication & Society, vol. 13 , pp. 289-308, Apr. 2010, doi:10.1080/13691180903456546.

[8] W3C. *W3C Mission*. 2012; Retrieved February 2014 from: http://www.w3.org/Consortium/mission.html

[9] W3C. *Web Content Accessibility Guidelines (WCAG) 2.0.* 2008; Retrieved March 2014 from: http://www.w3.org/TR/WCAG20/

[10] P. A. McLellan, "Web Accessibility," (Master's thesis). 2011; Available from ProQuest Dissertations and Theses database. (UMI No. 1493599).

[11] W3C. *Complete List of Web Accessibility Evaluation Tools.* 2006; Retrieved March 2014 from: http://www.w3.org/WAI/ER/tools/complete

[12] E. R. de Souzaand and C. Mont'Alvão, "Web accessibility: evaluation of a website with different semi-automatic evaluation tools," Work: A Journal of Prevention, Assessment and Rehabilitation, vol. 41, pp. 1567-1571, Feb. 2012, doi: 10.3233/WOR-2012-0354-1567.

[13] S. K. Kane, J. A. Shulman, T. J. Shockley and R. J. Ladner, "A web accessibility report card for top international university web sites," The 2007 international cross-disciplinary conference on Web accessibility (W4A). ACM, May 2007, pp. 148-156, ISBN: 1-59593-590-8

[14] M. Eyadat and J. Lew, "Web accessibility factor a key focus for serving students," Review of Business Research, vol. 11, pp. 29, October 2011.

[15] J. M. Fernández, J. Roig, and V. Soler, "Web Accessibility on Spanish Universities," Evolving Internet (Internet), IEEE, Sept. 2010, pp. 215-219, ISSN: 2156-7190, ISBN: 978-1-4244-8150-7

[16] B. J. Hashemian, "Analyzing web accessibility in Finnish higher education," ACM SIGACCESS Accessibility and Computing, Sept. 2011, pp. 8-16. ISSN: 1558-2337

[17] S. Kuakiatwong, "Evaluating Web Accessibility and Usability at Thailand Cyber University for Totally Blind Users," World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education, Oct. 2010, pp. 1904-1907. ISSN: 1880094835

[18] Webometrics. "*Webometrics Ranking : Ranking Web of Thailand Universities,"* Retrieved February 2014 from: http://www.webometrics.info/en/asia/thailand

[19] W3C. *Website Accessibility Conformance Evaluation Methodology 1.0.* 2012; Retrieved April 2014 from: http://www.w3.org/TR/WCAG-EM/

[20] S. Hackett and B. Parmanto, "Homepage not enough when evaluating web site accessibility," Internet Research, vol. 19, pp. 78-87, 2009.

[21] M. Vigo, J. Brown, and V. Conway, "Benchmarking web accessibility evaluation tools: measuring the harm of sole reliance on automated tests," The 10th International Cross-Disciplinary Conference on Web Accessibility, ACM, May 2013, pp. 1-10, doi:10.1145/2461121.2461124.

[22] C. Power, H. Petrie, A. P. Freire and D. Swallow, "Remote evaluation of WCAG 2.0 techniques by web users with visual disabilities," Universal Access in Human-Computer Interaction. Design for All and eInclusion, vol. 6765, pp. 285-294, 2011.

[23] K. A. Harper and J. DeWaters, "A quest for website accessibility in higher education institutions," The Internet and Higher Education, vol. 11, pp. 160-164, 2008.

[24] J. Lazar, P. Beere and K. D. Greenidge, "Web accessibility in the Mid-Atlantic United States: a study of 50 homepages," Universal Access in the Information Society, vol. 2, pp. 331-341, November 2003.

[25] T. Elias, "Universal instructional design principles for Moodle," International Review of Research in Open and Distance Learning, vol. 11, pp. 110-124, March 2010.

[26] P. Bohman, "University web accessibility policies: a bridge not quite far enough," World Conference on Educational Multimedia, Hypermedia and Telecommunications, 2004, pp. 5395-5400, ISBN: 1880094533

[27] D. A. Bradbard, C. Peters, and Y. Caneva, "Web accessibility policies at land-grant universities," The Internet and Higher Education, Association for the Advancement of Computing in Education (AACE), Jan. 2010, pp. 258-266, ISBN 978-1-880094-53-2

[28] Powermapper. *SortSite*. 2014; Retrieved February 2014 from:http://www.powermapper.com/products/sortsite/index.htm

[29] R. William, *Moodle 2.0 E-Learning Course Development.* 2011; Packt Publishing Ltd.

[30] R. Patrick, "Approximating the Shapiro-Wilk W-Test for non-normality," Statistics and Computing, 1992, pp. 117-119, ISBN: 0960-3174

 ISBN: 978-1-61208-367-4

A Usability Inspection Approach to Assist in the Software Development Process

Priscila Silva Fernandes, Bruno Araújo Bonifácio
Institute of Exact Sciences and Technology (ICET/UFAM)
Federal University of Amazonas
Itacoatiara, Brazil
{pry.bila, brunobonni}@gmail.com

Priscila Silva Fernandes, Tayana Uchôa Conte
Software Engineering and Usability Group (USES)
Institute of Computing (IComp)
Federal University of Amazonas
Manaus, Brazil
{priscila.fernandes, tayana}@icomp.ufam.edu.br

Abstract— **Several approaches have been proposed to ensure the quality of interactive systems. However, interactive systems continue to reach users with malfunctions, such as usability, communicability and interaction errors. Researches show that the lack of usability knowledge in software development organizations is an obstacle for usability evaluation. Our research goal is to popularize usability inspections so that even novice inspectors are able to perform it. Aiming to provide an approach to be used during the development process of web application, we have proposed the WE-QT technique. We are using an experimental methodology to evolve our technique and transfer it from the academy to industry. This paper presents a new comparative study; the results show that WE-QT technique is more efficient than and as effective as the compared technique.**

Keywords-usability inspection; novice inspectors; web application; experimental study.

I. INTRODUCTION

Interactive systems have been widely developed and used nowadays. Integrating this growth to web services, users are able to be more connected. Despite the great evolution of web applications, and the existence of various approaches addressing their development [1], users sometimes experience malfunctions when interacting with them. These malfunctions are generally caused by the poor interaction design [2].

Web applications with bad interaction design leads not only to users dissatisfaction and frustration, but also to rework during the development and maintenance phases, costs surpass, and market disadvantage [3]. Usability is a quality factor that can improve interaction design of software products [4][5]. Therefore, improving usability of web application can minimize users' interaction difficulty and improve the quality of these applications [6][7]. However, researches show that a large fraction of this problem is originated on the development process of these systems, which sometimes does not embody Human Computer Interaction (HCI) principles and methods for the development and evaluation of these applications [8][9][10]. Some authors [8][9][10] state that developers do not use, avoid or incorrectly apply HCI principles. This fact is due to the scarcity of knowledge and experience in concepts and practice of HCI area.

We proposed the Web Evaluation – Question Technique (WE-QT) [11][12] seeking to assist software developers performing usability inspections, and hence to improve the quality of the software products. The WE-QT technique is currently in the third version. We are using an experimental methodology to evaluate and evolve our technique [14]. This paper presents a comparative experimental study between the new version of WE-QT and its base technique, the Web Design Perspectives-based Inspection – Reading Technique (WDP-RT). Results show that our technique is as effective as and more efficient when compared to WDP-RT. Future work includes running studies with a major sample and comparing WE-QT to other methods.

The remaining of this paper is organized as follows. Section II addresses usability concepts and methods, along with the description of the new version of the WE-QT technique. Section III describes the experimental methodology and the results of the study. Section IV presents the conclusions.

II. USABILITY EVALUATION

According to Krug [15], users do not want to spend time trying to discover what the web application is about, figuring out whether an unusual button is actually a link, or how to go back to a previous visited page. Several researches propose approaches to ensure the quality of software product. Some propose tools and techniques specific to evaluate usability [6]. Usability evaluation methods can be divided into two groups: usability inspections and usability tests [16][18]. Usability inspection consists of a detailed interface analysis by an expert, while usability test seeks to uncover problems based on user observation [6]. Usability test is often more expensive because it requires users' time and specific material or infrastructure, such as usability labs [6]. Usability inspection was proposed as a better cost-effective method [6]. The majority of the researches focus on usability tests [8]. Our work, however, is centered on usability inspections. We also restricted the software products to web applications, and the target public to novice inspectors. Literature provides similar works [10][21][22][23][24][25][26][27]. Conte et al. [10] proposed the Web Design Perspective-Based Usability Evaluation Technique (WDP). The WDP is a checklist technique that uses the Nielsen's Heuristic Evaluation [2] illuminated by the three web-design perspective: Presentation, Conceptual and Navigation [10]. The authors

 ISBN: 978-1-61208-367-4

state that the technique is feasible, however inspectors had difficulties to apply the technique due the lack of knowledge and experience in usability and inspections [10]. Gomes *et al*. [27] propose the WDP-RT technique. The WDP-RT has the WDP technique as base and it is detailed in the next subsection.

A. *WDP-RT*

WDP-RT is a reading technique specific designed to inspectors with little knowledge on usability and inspections. The WDP-RT technique consists of a three pages document containing instructions to assist the evaluator uncovering usability problems. The reading approach provided by the WDP-RT technique does not simplify the inspections to novice inspectors and it is generally very time consuming, since inspectors have to read the three-page-document to carry out the inspection. The results of empirical studies to evaluate the WDP-RT technique indicated that the inspectors still have difficulty on understanding its instructions and applying it [28]. The WDP-RT technique is available at [33]. Both WDP and WDP-RT techniques require training on usability and on the technique before the inspection.

B. *The WE-QT Techqnique*

The WE-QT technique is an approach to guide novice evaluators performing usability inspections, and has the WDP-RT as base [11][12]. This research focused on novice inspectors aiming to reach industry workers that have little experience/knowledge on human-computer interaction concepts and practices, more specifically on usability and inspections. Our technique is composed by questions [12]. The questions lead the inspector through a flow that is adaptable by the elements present on the interface [14]. The mapping, provided by the question flow is illustrated in Figure 1.

The WE-QT technique is currently in its third version. Our technique does not require training on usability or on the technique itself before utilizing it. Since this approach can be applied by development team itself, reducing the need of executing usability tests or hiring an expert inspector, it is considered a cheaper option to improve the quality of web application. Some improvements made for the third version of the WE-QT technique are as follows: (1) Adding descriptions/examples to illustrate each question/affirmative, aiming to increase the information to assist the novice

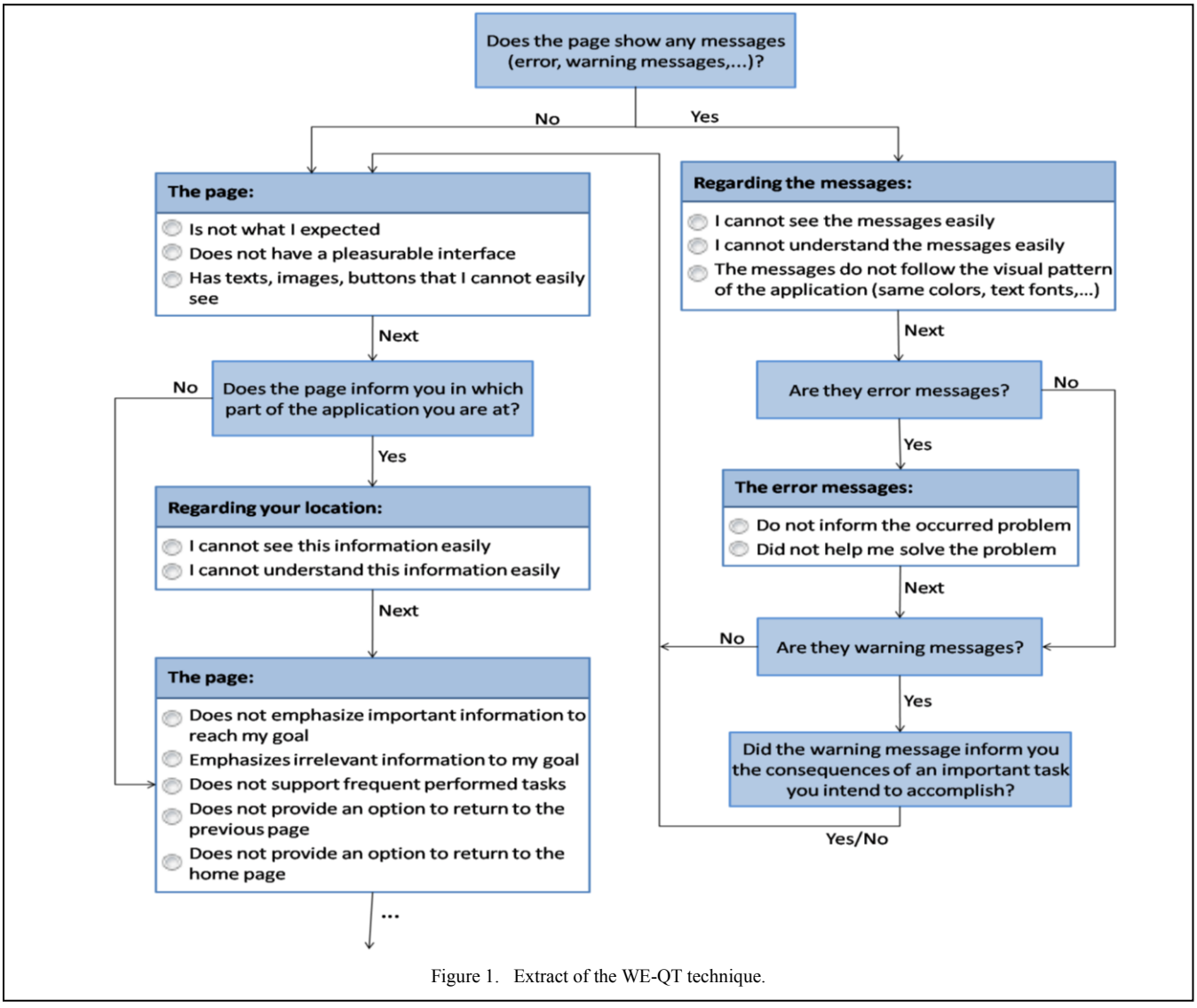


Figure 1. Extract of the WE-QT technique.

inspectors to better judge possible problems, Table I shows an extract of the descriptions and examples of the questions/affirmatives; (2) Implementing the scope of the inspected elements, we added questions addressing pop-ups and logins; (3) Adding initial instructions on how to conduct an inspection flow; (4) Relocating some questions, aiming to avoid mistakes concerning switching pages when the inspection flow is executed.

TABLE I. ILLUSTRATION OF THE EXAMPLES PROVIDED BY THE WE-QT TECHNIQUE – THIRD VERSION.

Question/Affirmative	Example
I cannot see the messages easily	If the message source is small, or if the message is in a difficult location to be seen
Does the page inform you in which part of the application you are at?	Some mechanism to inform you on which page of the system you are at
Is the page part of a page sequence of a task (e.g., a registration with several steps)?	A registration form with several steps, or a form with several pages
Are the mandatory fields to be filled in well defined?	Defined by a symbol as "*" or a similar one

III. EMPIRICAL EVALUATION

Following the experimental methodology, we designed a observational comparative study to evaluate the WE-QT technique. As the WE-QT has the WDP-RT as its base technique, we initially used the WDP-RT technique as the compared technique. We used statistical test to analyze the quantitative data and we present the subjective results. The second observational study is detailed below.

We selected 16 students from the third year of Information System course at the Federal University of Amazonas (UFAM), which were attending an Analysis and Design class. All the participants were familiar with Web applications. One participant had high knowledge and experience on usability, three participants had medium, four of them had low and the others had none. We divided them into two groups: Group 1 (participants used the WE-QT technique) and Group 2 (participants used the WDP-RT technique). All the participants carried out the inspection individually. Table II shows the participants characterization regarding usability knowledge and experience. The participants had also participated in another study, concerning usability of models, in which it was provided two hours of usability training.

The evaluation object was the MPS.Br Portal [20]. This application is responsible for providing information concerning the MPS.Br program. The MPS.Br is a nationwide program, equivalent to the CMMI [29], for software process improvement in Brazilian organizations. The MPS.Br aims to establish a feasible pathway for organizations to achieve benefits from implementing software process improvement at reasonable costs, especially small and medium-size enterprises [30]. It was the same application used in the feasibility study, as described in [11]. However, due to the large number of pages to be evaluated in the feasibility study, we only selected two tasks, with two web pages each, to be executed by the inspectors during the evaluation. The following tasks comprised the inspection's context: Obtain information about the Implementation Guides. These guides describe orientations on how to implement some expected results of the MPS.Br program and access the presentations provided by the MPS.Br Portal, such as presentations about the MPs.Br program, workshops, and projects.

We used Morae (version 3.3) usability testing software to capture the inspection section of each inspector and to assist the collection of the perceptions of each inspector during the evaluation. Subjective data was gathered at the completion of the inspection phase using post-inspection questionnaires. We provided the subjects with the Inspection Guide and a Consent Form (all the subjects signed the consent form before starting the inspection).

To support the mapping process of the WE-QT technique, we developed an automated tool called WE-QT Assistant. The support tool was designed to minimize the effort of the inspectors during the problem detection phase. Therefore, the tool was developed to be located at the left side of the screen, allowing the inspection to be performed without the need to switch windows. The left side of Figure 2 illustrates the WE-QT Assistant. The Assistant provides text boxes in order to allow the inspectors describing the identified usability problems instead of needing an extra document to report the problems.

TABLE II. SUMMARY OF INSPECTION RESULTS PER SUBJECT

	Nº	Usabil. Exp.	Discrep.	Nº Real Prob.	False-positives	Time (min)	Effectiv. (%)	Effic. (Prob. /hour)
WE-QT	1	None	9	7	2	37	7,22%	11,35
	2	Low	17	9	8	37	9,28%	14,59
	3	Low	26	16	10	35	16,49%	27,43
	4	Medium	28	14	14	22	14,43%	38,18
	5	None	37	31	6	72	31,96%	25,83
	6	None	65	55	10	75	56,70%	44,00
	7	Medium	31	25	6	39	25,77%	38,46
	8	None	30	22	8	53	22,68%	24,91
WDP-RT	9	None	13	7	6	80	8,64%	5,25
	10	Low	21	9	12	86	11,11%	6,28
	11	Medium	43	24	19	109	29,63%	13,21
	12	High	19	9	10	65	11,11%	8,31
	13	Low	65	44	21	163	54,32%	16,20
	14	None	24	15	9	104	18,52%	8,65
	15	None	31	21	10	107	25,93%	11,78
	16	None	13	6	7	77	7,41%	4,68

In order to minimize the threats to validity, we developed a tool to support the WDP-RT technique as well. The tool is similar to the WE-QT Assistant, and also has text boxes to problem description and it is located on the left side of the screen.

 ISBN: 978-1-61208-367-4

In this study, we used the effectiveness and efficiency indicators. These indicators have been employed in other studies concerning usability inspection methods as well [10][11][19]. Effectiveness is defined as the ratio between the number of detected problems and the total of existing problems; and Efficiency is defined as the ratio between the number of detected problems and the time spent in the inspection.

We also collected participants' subjective impressions of the techniques through a post-inspection questionnaire, which was based on the Technology Acceptance Model (TAM) [31]. This model aims to examine a new technology usage and verify the user perceptions concerning usefulness and ease-of-use, the key determinants of individual technology adoption.

The experiment was used to test the following hypotheses concerning to the effectiveness and efficiency (null and corresponding alternative hypotheses are given):

- H01: There is no difference between the effectiveness of techniques WE-QT and WDP-RT.
- HA1: The effectiveness of the WE-QT technique is greater than the effectiveness of the WDP-RT technique.
- H02: There is no difference between the efficiency of techniques WE-QT and WDP-RT.
- HA2: The efficiency of the WE-QT technique is greater than the efficiency of the WDP-RT technique.

The inspection phase was carried out by each subject individually. They were provided with the instruments to accomplish the inspection and received instructions about the evaluation by the moderator. Subjects from Group 1 (WE-QT) were provided with a three minutes presentation on how to conduct an inspection flow, while subjects of Group 2 (WDP-RT) were provided with information concerning the

Figure 2. Subject using the WE-QT technique during the study.

inspection flow plus a five minutes exemplification of usability problems detection using the WDP-RT technique and its instructions. It is worth to mention that the subjects from Group 1 did not receive training on the WE-QT technique. Once the inspector understood the procedures, the inspection process began. Figure 2 shows a subject from Group 1 evaluating the MPS.Br Portal with the WE-QT Assistant (WE-QT technique, problem detection and description); it also illustrates in the interface of the application an example of an uncovered problem. One researcher acted as the facilitator, being responsible for conducting the detection phase and passing the initial information to the subjects. After the detection phase, the subjects received the post-inspection questionnaire by e-mail and they could answer them at home.

At the end of the inspection phase, the researches elaborated a list containing all usability problems detected by the inspectors, without duplicates. Then, a meeting attended by the researchers and a control group formed by usability specialists took place. The list of problems was discriminated to classify these problems into real problems or false positives. To eliminate any possible influence during the discrimination meeting, the problem list did not contain any information about witch technique uncovered witch problems. The authors of the technique did not influence the discrimination, they were not allowed to comment or give

 ISBN: 978-1-61208-367-4

their opinion about whether the discriminated problems were real problems or false-positives.

A. Results

After the discrimination meeting, we were able to analyze the gathered data. We computed the number of detected problems, false-positives, time spent during the inspection phase, efficiency and efficacy for each inspector of each group.

1) Quantitative results: As a result of the inspection, the inspectors identified a total of 135 real problems, including both techniques. The WE-QT group detected a total of 97 problems, while the WDP-RT group uncovered 81. Table III shows the averages for the time, and effectiveness and efficiency indicators. Regarding the efficiency indicator, inspectors detected an average of 31.91 defects per hour using the WE-QT technique. Table II presents the overall result of the usability evaluation for each inspector, including their experience level.

TABLE III. AVERAGE EFFECTIVENESS AND EFFICIENCY PER GROUP.

Technique	Average Effectiveness (%)	Average Efficiency (Prob/hour)	Average Time (min)	Total Known Defects
WE-QT	23,07	28,09	46,25	97
WDP-RT	20,83	9,29	98,88	81

We carried out a statistical analysis using the statistical tool SPSS (SPSS Statistics version 17.0), and $\alpha = 0.05$. This choice of statistical significance was motivated by the small sample size used in this experiment.

Concerning H1: Effectiveness of Techniques WE-QT and WDP-RT, we compared the two samples, Group 1 (WE-QT) and Group 2 (WDP-RT), using the Mann-Whitney test, a non-parametric test, we found no significant differences between the two groups ($p = 0.753$). These results show that both techniques provided similar effectiveness when used to inspect the MPS.Br Portal. Figure 3 shows the boxplot with the distribution of effectiveness per subject, per technique.

Regarding H2: Efficiency of Techniques WE-QT and WDP-RT, the boxplots with the distribution of efficiency per subject, per technique (see Figure 4) show that Group 1 (WE-QT) was considerably more efficient than Group 2 (WDP-RT) to inspect the usability of the MPS.Br Portal: Group 3's median is significantly higher than Group 2's. When we compared the two samples using the Mann-Whitney test, it confirmed significant statistical differences between the two groups ($p = 0.021$), which supports the alternative hypothesis HA2, and therefore rejects the null hypothesis H02. These results suggest that the WE-QT technique efficiency was significantly higher than the WDP-RT's. Results show that effectiveness of both techniques is similar; however the WE-QT technique was nearly three times more efficient then the WDP-RT technique.

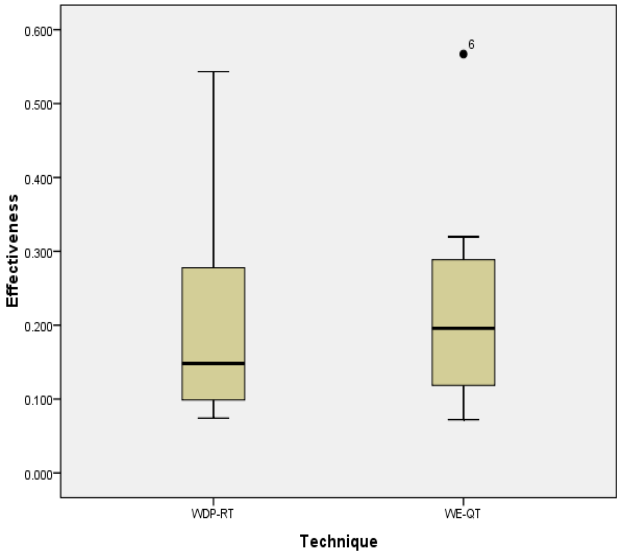


Figure 3. Boxplots for number of defects found per subject per technique.

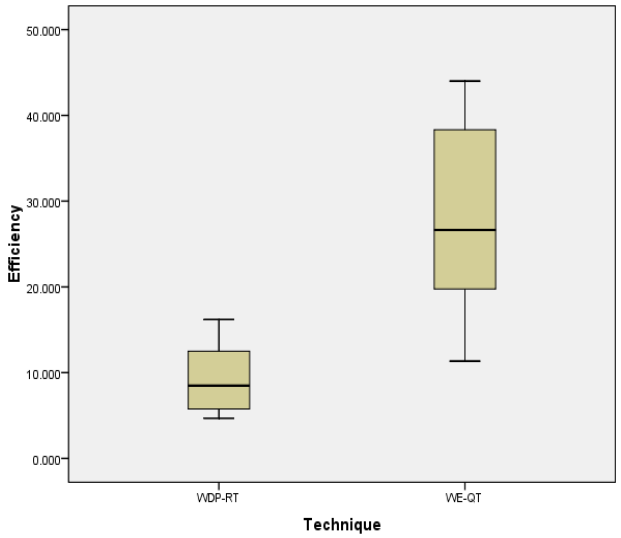


Figure 4. Boxplots for efficiency per subject per technique.

2) Subjective results: When we proposed the WE-QT technique, one of our aims was to improve users' satisfaction when using the technique, therefore we are also evaluate this aspect in this study. We collected the subjects' opinions with respect to key determinants of individual technology adoption, perceived ease of use and usefulness; collected with the post-inspection questionnaire, based on the TAM model. The questionnaire had closed and opened questions. The closed questions were based on a 6 value scale – 0% (Totally Disagree), 1%-30% (Strongly Disagree), 31%-50% (Partially disagree), 51%-69% (Partially Agree), 70%-99% (Strongly Agree) and 100% (Totally Agree); note that it did not have a neutral option, forcing the subjects to stand a position on whether they agree or disagree. Figure 5 shows the average subject ratings, together with standard deviations. The ease of use

 ISBN: 978-1-61208-367-4

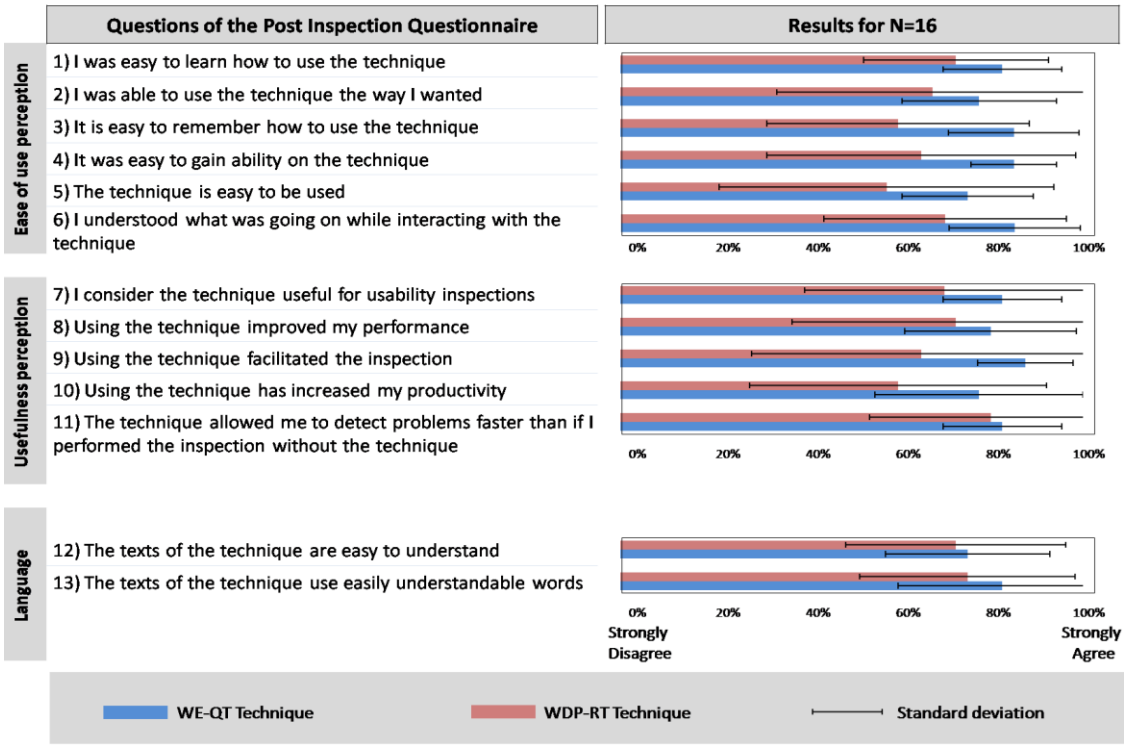


Figure 5. Results of the post-inspection questionnaire.

perception comprises factors such as learnability, customization of use, ability gain and understanding of the technique; while usefulness perception covers factors such as usefulness, performance improvement, productivity, and efficiency when using the technique. We also add questions addressing the language of the techniques, to identify any possible improvements suggestions. The questionnaire contained discursive questions as well. According to Figure 5, the WE-QT technique was perceived slightly more easy to use and useful then the WDP-RT technique. The both techniques were ranked similarly to the language aspects. The participants' subjective answers could be affected by the tool to support the techniques automations, which we will evaluate in further studies. The subjective data were important to improve the WE-QT technique.

IV. CONCLUSION AND FUTURE WORK

In this paper, we evaluated the WE-QT technique, a question-based usability evaluation technique for web applications, specifically tailored for software developers with little knowledge HCI principles and concepts, more specifically on inspections and usability. We used a formal statistic experiment to compare the efficiency and effectiveness of both techniques: WE-QT and WDP-RT. The results showed that our technique in significantly more efficient than and as effective as the WDP-RT technique. We also evaluate the techniques concerning the perceived ease of use, usefulness and language of the techniques. Subjective results showed that our technique was perceived slightly more easy to use and useful then the WDP-RT technique. These results are very promising. However, we will continue to research our technique. Limitations of this research include: focusing on novice inspectors, additional research is required to specific address this topic; the small sample of participants; comparing WE-QT only to one technique. Future work includes: (1) improvement of the technique based on a detailed analysis of the detected usability problems, false-positives and time spend; (2) investigation of a new arrangements for each usability questions, for instance, if efficiency, effectiveness and user satisfaction can be improved if the questions regarding the web application as a whole came first then the questions regarding each individual page; (3) further studies comparing the WE-QT technique with other usability inspection techniques specific for evaluate web applications, with a greater number of subjects; and (4) the replication of the experiment in an industrial environment. With this research we also aim to encourage professionals involved on the development process of interactive systems, such as developers, analysts, testers and stakeholders, to use HCI principles and methods in the development cycle.

ACKNOWLEDGMENT

The authors thank the support granted by CAPES process 00.889.834/0001-08; the researchers Luis Rivero and Natasha Valentim for participating on the discrimination meeting; Martha Fernandes and also all participants of the conducted study.

REFERENCES

[1] M. Zaki and P. Forbrig, "A Methodology for Generating an Assistive System for Smart Environments Based on Contextual Activity Patterns" The Fifth ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2013) ACM, Jun. 2013, pp. 75-80, ISBN: 978-1-4503-2138-9, doi: 10.1145/2494603.2480327.

[2] J. Nielsen, Heuristic evaluation. Jakob Nielsen, Mack, R. L. (eds), "Usability inspection methods", Heurisitic Evaluation, John Wiley & Sons, Inc, 1994.

[3] P. Lew, L. Zhang, and S. Wang, "Model and measurement for web application usability from an end user perspective". Quality Assessment in Web, Jun. 2009, pp. 1613-0073.

[4] E. Mendes, N. Mosley, and S. Counsell, "The need for web engineering: an introduction". Mendes E., Mosley N. (eds.): 'Web engineering' Springer, pp. 1-26, 2006.

[5] ISO (1997), ISO 9241-11: Ergonomic requirements for office work with visual display terminals (VDTs). Part 11 — Guidelines for specifying and measuring usability. Gènève: International Organisation for Standardisation.

[6] M. Matera, F. Rizzo, and G. Carughi, "Web Usability: Principles and Evaluation Methods". E. Mendes, N. Mosley, (eds), Web Engineering, Spinger Verlag, 2006.

[7] B. Bonifácio, D. Santos, C. Araújo, S. Vieira, and T. Conte, "Applying Usability Inspection Techniques for Evaluating Web Mobile Applications" 9th Brazilian Symposium of Human Factors on Computer Systems (IHC 2010), Oct. 2010, pp. 189-192,. (In Portuguese).

[8] J. Bak, K. Nguyen, P. Risgaard, and J. Stage, "Obstacles to Usability Evaluation in Practice: A Survey of Software Development Organizations" Nordic conference on Human-computer interaction (NordiCHI 2008), Oct. 2008, pp. 23 – 32.

[9] B. Bonifácio, P. Fernandes, H. Oliveira, and T. Conte, "UBICUA: A Customizable Usability Inspection Approach for Web Mobile Applications" IADIS International Conference WWW/Internet, vol. 1, Nov. 2011, pp. 45-52.

[10] T. Conte, J. Massolar, E. Mendes, and G. Travassos, "Web Usability Inspection Technique Based on Design Perspectives" IET Software Journal, vol. 3, n. 2, Apr. 2009, pp. 106-123,.

[11] P. Fernandes, L. Rivero, B. Bonifácio, D. Santos, and T. Conte, "Evaluating a New Usability Inspection Approach: a Quantitative and Qualitative Analysis" 8th Experimental Software Engineering Latin American Workshop (ESELAW 2011) vol. 1, Apr. 2011, pp. 67-76. (In Portuguese).

[12] P. Fernandes, B. Bonifácio, and T. Conte, "Improving a Web Usability Inspection Technique through an Observational Study" 24th International Conference on Software Engeneering and Knowledge Engineering, vol.1, Jul. 2012, pp. 588-593.

[13] F. Shull, J. Carver, and G. Travassos, "An empirical methodology for introducing software processes" ACM SIGSOFT Software Engineering Notes, vol. 26, n. 5, Sep. 2001, pp. 288-296, doi: 10.1145/503209.503248.

[14] P. Fernandes, B. Bonifacio, and T. Conte, "WE-QT: A Web Usability Inspection Technique to Support Novice Inspectors" 21th Brazilian Symposium on Software Engineering (SBES 2012), Sep. 2012, pp. 11-20, ISBN: 978-1-4673-4472-2, doi: 10.1109/SBES.2012.30.

[15] S. Krug, Don't Make Me Think: A Common Sense Approach to the Web (2nd Edition), 2005.

[16] J. Offutt, "Quality Attributes of Web Software Applications" IEEE Software, vol. 19, n. 2, Mar.-Apr. 2002, pp. 25-32.

[17] E. Luna, J. Panach, J. Grigera, G. Rossi, and O. Pastor, "Incorporating usability requirements in a test/model-driven web engineering approach" Journal of Web Engineering, vol. 9, n. 2, Mar. 2010, pp. 132-156.

[18] H. Rocha and M. Baranauskas, "Design and Evaluation of Human-Computer Interface" M.C.C., 1. ed. Campinas: Emopi Publisher and Graphic, vol. 1, p. 244, 2003. (in Portuguese).

[19] A. Fernandez, S. Abrah, and E. Insfran, "Towards to the validation of a usability evaluation method for model-driven web development" 4th International Symposium on Empirical Software Engineering and Measurement, Sep. 2010, pp. 54, ISBN: 978-1-4503-0039-1 doi: 10.1145/1852786.1852855.

[20] MPS.Br: Program for software process improvement in Brazil (in Portuguese) http://www.softex.br/mpsbr/ [retrieved: January, 2013].

[21] M. Costabile and M. Matera, "Guidelines for Hypermedia Usability Inspection" IEEE Computer Society Press, vol. 8, n. 1, pp. 66-69, 2001.

[22] A. Strauss and J. Corbin, "Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory", 2ed., SAGE Publications, 2010, pp. 1-3, ISSN: 0318-9090.

[23] L. Triacca, A. Inversini, and D. Bolchini, "Evaluating Web usability with MiLE+" 7th IEEE International Symposium on Web Site Evolution, Sep. 2005, pp. 22-29, ISBN:0-7695-2470-2, doi: 10.1109/WSE.2005.6.

[24] M. Blackmon, P. Polson, and M. Kitajima, "Cognitive walkthrough for the web" Conference on Human Factors in Computing Systems (CHI 2002) ACM, Apr. 2002, pp. 463–470, ISBN:1-58113-453-3 doi:10.1145/503376.503459.

[25] L. Filgueiras, S. Martins, C. Tambascia, and R. Duarte, "Recoverability Walkthrough: An Alternative to Evaluate Digital Inclusion Interfaces" Latin American Web Congress, Nov. 2009, pp. 71-76, ISBN: 978-0-7695-3856-3, doi: 10.1109/LA-WEB.2009.32

[26] M. Allen, L. Currie, S. Bakken, V. Patel, and J. Cimino, "Heuristic evaluation of paper-based Web pages: A simplified inspection usability methodology" Journal of Biomedical Informatics, vol. 39, n. 4, Aug. 2006, pp. 412 – 423, doi: http://dx.doi.org/10.1016/j.jbi.2005.10.004.

[27] M. Gomes *et al.*, "WDP-RT: A usability inspection reading technique for web applications" 6th Experimental Software Engineering Latin American Workshop (ESELAW 2009), vol. 1, Nov. 2009, pp. 124 – 133. (In Portuguese).

[28] M. Gomes, F. Santos, D. Santos, G. Travassos, and T. Conte, "Evolving a Usability Evaluation Technique through In Vitro and In Vivo Studies" 9th Brazilian Symposium on Software Quality (SBQS 2010), vol. 1, Jun. 2010, pp. 229 – 244. (In Portuguese)

[29] CMMI - Capability Maturity Model Integration. http://www.sei.cmu.edu/cmmi/. [retrieved: July, 2014].

[30] M. Montoni, A. Rocha, and K. Weber, "MPS.BR: a successful program for software process improvement in Brazil". Software Process Improvement and Practice, vol. 14, Sep. – Oct. 2009, pp. 289-300, doi: 10.1002/spip.428.

[31] F. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology", MIS Quarterly, vol. 13, n. 3, Sep. 1989, pp. 319-340.

[32] A. Fernandez, E. Insfran, and S. Abrahão, "Usability evaluation methods for the web: A systematic mapping study" Journal Information and Software Technology, vol. 53, n. 8, Mar. 2011, pp. 789-817, doi: 10.1016/j.infsof.2011.02.007.

[33] M. Gomes and T. Conte, "WDP-RT v2: A reading technique for usability inspection of Web applications (In Portuguese)". Technical Report RT-DCC-ES002/2009, DCC/UFAM, 2009.

 ISBN: 978-1-61208-367-4