# 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.

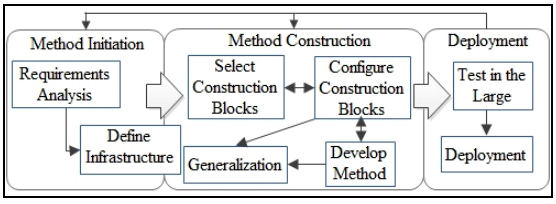


Figure 1. Generic SME Process Framework – Adapted from [13].

 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.

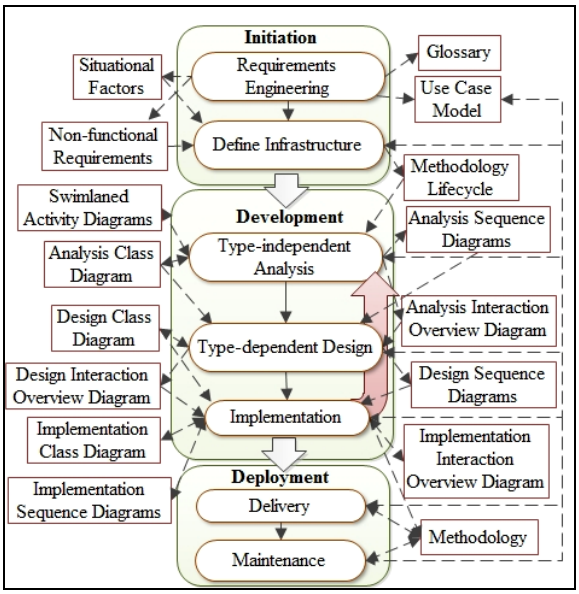


Figure 2. UCDMD Process.

*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 |

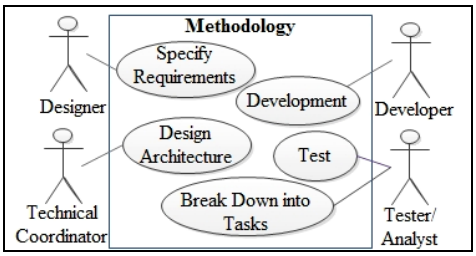


Figure 3. Example of a use case diagram for methodology development.

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:**<br>- The requirements of the current iteration have been determined. |
| **Main flow:**<br>1. The use case is started when the Analyst requests the requirements of the current iteration to break them down into fine-grained tasks.<br>2. Methodology instructs Analyst on how to break down requirements.<br>3. For each requirement of this iteration:<br>3.1. Analyst breaks down requirement.<br>3.2. Methodology instructs Analyst on how to store the tasks.<br>3.3. Analyst stores the tasks.<br>3.4. Methodology instructs Analyst on how to evaluate the results.<br>3.5. Analyst evaluates the results. |
| **Postconditions:** Fine-grained tasks are ready for the current iteration. |
| **Alternative Flows:**<br>- Suspend breaking down into tasks. |

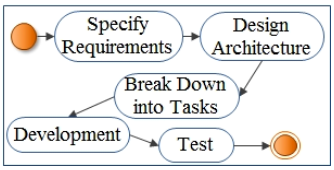


Figure 4. Example of an interaction overview diagram.

 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.

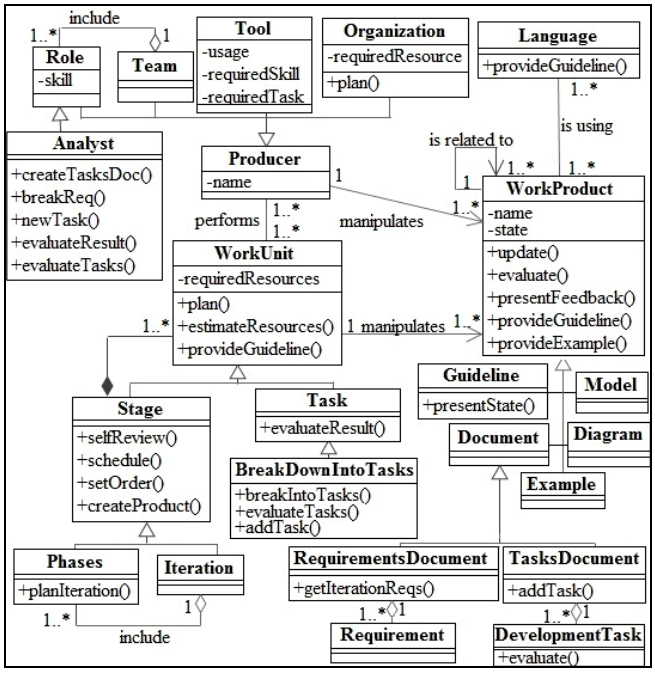


Figure 5. Example of a design class diagram.

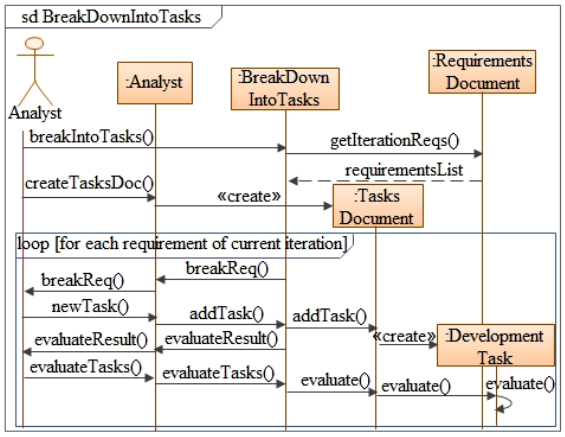


Figure 6. Example of a design sequence diagram.

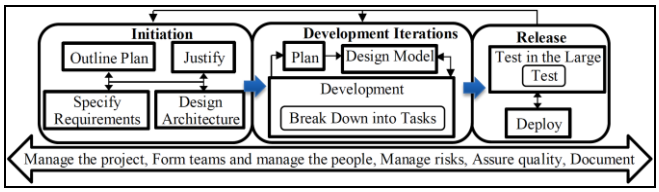


Figure 7. Example of an implemented methodology (lifecycle view).

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

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* |