Towards the Development of Integrated Reuse Environments for UML Artifacts

Moataz A. Ahmed
Information and Computer Science Department
King Fahd University of Petroleum and Minerals
Dhahran 31261, Saudi Arabia
moataz@kfupm.edu.sa

***Abstract*—Systematic software reuse is recognized to achieve better software, faster and at a lower cost. The benefits of reuse can be maximized if types of early stage software artifacts can be easily reused. In early-stage reuse, once a match is found, all related later stages artifacts for the match can also be reused. However, the development of integrated reuse environments to allow managing and reusing repositories of early stage artifacts has not caught adequate attention of researchers yet. In response to this problem, we propose an approach to the development of environments integrated with CASE tools and capable facilitating early-stage artifacts reuse. Successful implementation of such environments is expected to improve the software quality and developers productivity.**

Keywords-early-stage artifacts; design reuse; integrated reuse environment; similarity metrics; multi-view similarity.

I. INTRODUCTION

Systematic software reuse has clear benefits to include reduction in overall development costs, increased reliability, reduced process risk, effective use of specialists, standards compliance, and accelerated development [1]. Features of the object-oriented (OO) software development paradigm, such as abstraction and encapsulation, encourage reuse of software by enabling building reusable blocks of *code*. However, it has been recognized for long that reuse of *early-stage* artifacts are particularly more beneficial than reuse of later-stage artifacts [10]. Types of early-stage reusable artifacts include [2]:

Domain Models: These can be reused at the earliest stage of the software development process, the domain analysis stage. Very few systems exist that exploits the reuse of artifacts at this stage. An example of such a system is the work of Blok and Cybulski [3]. Another example is the generic application frames in the ITHACA development environment [4]. Yet, a more recent example can be found in the software product lines approach, which was often touted as a silver bullet for actualizing software reuse goals [5][7][9].

Requirement Specifications: These artifacts can be reused during the requirements analysis phase of the software lifecycle [10]. An example of how a requirements specification reuse may be assisted by a software tool is described by Cybulski and Reed's [11].

Design: These artifacts can be reused during the design phase. An example of a design repository is the SPOOL Design Repository [12]. Another example is the work of Lee and Harandi [13].

In early-stage reuse, once a match is found, all related later stages artifacts for the match can also be reused. For instance, if an analysis model for a previous project is found to match the analysis model of a current project, then the previous project's design, code, test data, and relevant documentation may be reused in the current project.

Early-stage artifacts reuse, despite its clear benefits, suffers from a few problems though. Reuse problems include increased maintenance costs, the not-invented-here syndrome, *lack of tool support*, *difficulty of maintaining a library of reusable artifacts*, and *the cost of locating and adapting reusable artifacts* [1].

A step towards a solution to the problems above could be the development of effective tightly-knit tools to allow finding and reusing exiting design artifacts and what follows based on matching requirements specification. For maximal utilization within the day-to-day activities, such tools should be offered through a reuse environment integrated with some prominent CASE tools; hence, Integrated Reuse Environment (IRE). In this paper, we present an approach to the development of IREs to maximize the designer's productivity. The approach will be focusing on reusing Unified Modeling Language (UML) artifacts. The rationale behind focusing on UML is that it is considered the de-facto standard for expressing early-stage OO artifacts (e.g., analysis and design models) [8]. Accordingly, tools and techniques to support reusing UML artifacts would facilitate and encourage more early-stage reuse. However, to the best of our knowledge, there is IREs that allow finding and reusing exiting UML design artifacts and what follows based on matching requirements specification.

The rest of the paper is organized as follows. Section II presents a framework for assessing the similarity between UML artifacts of different projects. Section III discusses related work. Section IV lists some research questions to be answer in future work in order to realize effective IREs. Section V concludes the paper.

II. MULTI-VIEW UML MODELS SIMILARITY

During the requirement-analysis phase of the software development life cycle, the system requirements are typically modeled and analyzed from related but different viewpoints where each view represents one aspect of the software system to be developed. The division into different views is arbitrary and typically includes at least three views namely *structural view*, *functional view*, and *behavioral view* [21][22][23][24], each capturing important

 ISBN: 978-1-61208-165-6

aspects of the system, but all required for a complete description of the system. One or more kinds of diagrams provide a visual notation for the concepts in each view. A typical software procedure incorporates all three aspects [21]. Models from different views are meant to be compared to discover requirements that would be missed using a single view [1][22]. Structure describes static objects relevant to the domain in question, their relationships, attributes and their possible states; functions describe the input-output transformations, and behavior the instantiation and dynamics of the transformations with time.

It is worth noting here, though, that there is a general paucity of concepts for specifying the *functionality* of object communities [22]. The UML taxonomy of diagrams provides a logical organization for the various major kinds of diagrams into only two major categories: structure and behavior; with no category to represent the functional aspect [39]. Nevertheless, *use cases* can be interpreted as one means of specifying functionality, as according to Jacobson et al. [16], *they define the functionality inside the system and constitute a specific way of using some part of this functionality*. Clearly, a use case has also a flavor of behavior abstraction, as it is a special sequence of related transactions in the interaction between the actor and the system [22].

During the requirements engineering phase, the view-points analysis technique relies on these multi-view models where they are compared to discover requirements that would be missed using a single view.

We propose that developing IREs to allow early-stage OO artifacts reuse would require a framework of consistent multi-view similarity metrics that considers similarity across the three system views: functional view, structural view and behavioral view. For effective reuse of available designs of completed projects, the IRE should facilitate assessing the combined similarity between new requirements to the requirements of completed projects to provide closest match so that their design counterparts can be reused with minimal effort.

Considering UML, as the de-facto standard, we consider *Use Cases* as representative of the functionality (i.e., the services) that users require of the object oriented system. Use cases describe the typical interactions between the users of a system and the system itself, providing a narrative of how a system is used. During the requirements phase of a software project, analysts can take use cases to the next level by providing a more formal level of refinement in a form of *sequence diagrams*. Each use case is realized by one or more sequence diagrams that depict how the objects interact and work together to provide services. We propose considering the development of sequence diagrams similarity metrics in the functional view.

UML Structure diagrams show the static structure of the objects in a system. Examples of UML structure diagrams include the Class Diagram, Component Diagram, Object Diagram, Deployment Diagram, Package Diagram, Composite Structure Diagram, and the Profile Diagram. However, most of these diagrams are mainly used during the architecture and design phase to express artifacts at different design levels. During the requirements engineering phase, the static structure of the system is mainly captured using instances of the Class Diagram and Object Diagram. The Class Diagram shows the building blocks of any object-oriented system: the classes that make up a system. The potential for collaboration among these classes, through message passing, is shown in the relationship between these classes. Object diagrams show instances instead of classes. They are useful for explaining small pieces of class diagrams with complicated relationships, especially recursive relationships. We propose considering the development of class diagrams similarity metrics in the structural view.

Behavior diagrams can be used at two different levels: system level and object level. At the system level, behavior diagrams (mainly the State Machine Diagram, which is an object-based variant of Harel's statecharts [39]) is used to show the system behavior in response to user actions, as in user interface design [38]. At the object level, they show the dynamic behavior of the objects, including their methods, collaborations, activities, and state histories. The dynamic behavior of a system can be described as a series of changes to the system over time. Examples of UML behavior diagrams include Activity Diagram, Interaction Diagram, and State Machine Diagram. However, most of these diagrams are mainly used during the architecture and design phase to express artifacts at different design phases. During the requirements engineering phase, the system-level behavior is if interest and mainly captured using the State Machine Diagrams [38]. They are used to more formally describe the flows within or between use cases. We propose considering the development of state machine diagrams similarity metrics in the behavioral view.

The left-hand side part of Fig. 1 shows that design reuse is achieved by comparing the requirements of the new system to requirements of existing systems in a repository. The comparison is meant to assess the level of similarity. If the level of similarity between the best matching old requirements and the given new requirements is greater than a given threshold, corresponding design artifacts can be reused as a starting point for the new requirements. The right-hand side of the figure shows that the similarity assessment between the new and old requirements should consider the three views (depicted as dimensions): use cases representing the functional view, class diagrams representing the structural view, and state machine diagrams representing the behavioral view. It is worth noting here that sequence diagrams are used in the view-points analysis technique during the requirements engineering phase to double check the consistency between the structural view and the functional view as shown in the figure. Moreover, as stated above, sequence diagrams can be used to provide a more formal level of refinement of use cases. Accordingly, sequences diagrams could be used as a better representative of the functional view.

 ISBN: 978-1-61208-165-6

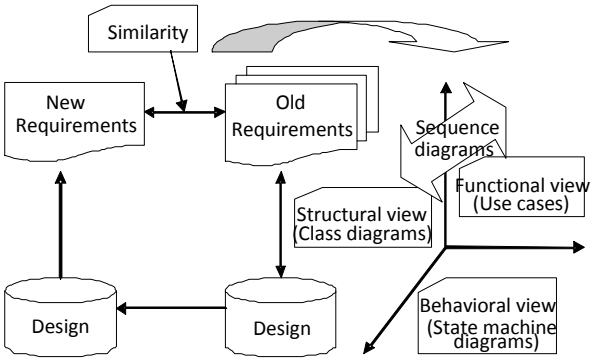

Figure 1. Various UML views in the context of similarity measurement.

In the sequel, we give a literature survey to identify the technology gap and corresponding research questions.

III. RELATED WORK

Developing with reuse consists of the following activities [15]: locating reusable artifacts (retrieval), assessing their relevance to current needs (assessment), and adapting them to those needs (adaptation). Locating reusable artifacts often involves some form of comparison of a query with candidate models in the repository. In every search, a *search space*, a *search goal*, and a *comparison function* are always defined. In software retrieval, the search space is known as the *software repository*. The search goal is called the *query*. The comparison function is called a *similarity metric*. How well the retrieved artifacts match the query depends on the soundness of the *similarity metric*.

The following subsection presents a literature review on reuse environments and similarity metrics.

A. Reuse Environments

Braga et al. [9] present the odyssey environment to support reuse-based software development environment based on component based software development. Odyssey is a framework where conceptual models, architectural models and implementation models are specified for pre-selected application domains. Similarly, eColabra [14] is a reuse environment that takes advantage of highly precise information retrieval techniques and graph visualization techniques to customize reuse during the classification and retrieval stages. Furthermore, eColabra applies information retrieval techniques to object oriented resources while exploiting the object oriented languages semantics and characteristics.

Correa et al. [18] present an approach to object oriented design reuse by integrating design patterns and anti-patterns into an object oriented design workbench. This allows the reuse of knowledge about good and bad object oriented design practices. Furthermore, a tool (OOPDTOOL) was developed to support the approach.

Cybulski et al. [11] combine keyword-based and faceted classifications of requirements and design. The keywords are extracted from the body of requirements text and are then translated into design terms of a faceted classification. Facets are subsequently used to determine affinity between requirements and design artifacts, which are used for reuse based refinement of requirements documents. Beyer et al. [19] present a success story in establishing an architecture-centric approach at a small development organization. They evolved the development organization towards systematic reuse by introducing an architecture-centric strategy for product development.

Recently, Martins et al. [20] have applied data mining techniques improve the search of reusable assets. They have applied association rules and clustering techniques to aid the knowledge extraction. They used the concept of log files to extract a historic pattern and facilitate the overall search process.

COTS-aware requirement engineering (CARE) [35] approach for component identification has the focus on the utility of knowledge base. The goals and requirements are specified as enterprise goals which are further sub specialized into component goals. CARE points out the importance to keep requirements flexible as they have to be constrained by the capabilities of available components. In this approach, requirements are classified as native requirements acquired from customers and foreign requirements of the COTS components. The method puts emphases on narrowing the gap between customer and component requirements by using knowledge base. The process model describes the activities performed to define the system agents, goals, system requirements, software requirements and architecture. The product model describes the format of the product created using the process. The meta-model describes the knowledge content and structure for the CARE approach. The method highlights the importance of mapping system requirements and product specification; however, it does not support the possible mismatch between both specifications.

The COTS usage risk evaluation (CURE) [36] is a 'front-end' analysis tool that predicts the areas where the use of COTS products will have the greatest impact on the program. CURE is designed to find risks relating to the use of COTS products and report those risks back to the organization. Ideally, CURE is performed on both the acquiring and the contracting organization, but this is not necessary. The evaluation consists of four activities: preliminary data gathering, on-site interview, analysis of data and presentation of results. The CURE method has proven to be a useful tool for organizations that acquire or develop COTS-based systems. However, there are several limitations of the current version of CURE ranging from considerable amount of manual analytical work performed by evaluators and training required by the evaluators.

B. Similarity Metrics

Retrieval is one of the activities in a *software reuse process*, which takes in a *query* as input and returns *reusable artifacts* (or objects of reuse) as output. Because the goal of software retrieval is to return most similar reusable software artifacts, we propose considering a

 ISBN: 978-1-61208-165-6

framework for multi-vie similarity assessment as discussed above.

In the sequel, we discuss related works in a chronological order starting with the latest. We conclude with a summary table (TBALE I) focusing on five aspects namely *artifacts considered* for reuse, artifact *internal representation*, *criteria for matching* artifacts (syntactic vs. semantic), use of extra-artifact *annotations* to guide matching, and the *search algorithm* used.

Ahmed [6] proposed a similarity metric to measure the similarity of UML models from the functional view using sequence diagrams. He used heuristic search techniques such as Genetic and Greedy Algorithms to assess similarities. His work did not consider consistencies with other views though.

Rufai [2] proposed a set of structural similarity metrics to measure the similarity in structural view for UML models. Different metrics capture the different structural aspects of the UML model.

William Robinson and Woo [25][26] present an automatic technique that provides assistance for use cases reuse. Given an initial description of the use case by software analyst, an automated graph based relational learner retrieves a set of similar use cases from a database. Their work uses automated graph based relational learner called SUBDUE. It represents an interaction diagram (sequence diagram) which provides a semantic structure that includes objects and methods of a use case as labeled graph that consists of vertices and edges. It uses a structure similarity measure to determine the distance between query structure and the structures available in repository. The technique does not make use of extra annotations and can be incorporated into tools like rational rose. In their other work, the authors have developed a CASE tool called REUSER which uses SUBDUE algorithm to automatically retrieve related UML sequence diagrams for reuse.

Saeki [27] has made an investigation into which parts of a use case description can be catalogued as reusable patterns and template for requirement analysis process. He listed the following parts as candidates for reuse:

- Use case templates for describing use cases.
- Use case patterns for providing the reusable and changeable structures for use cases.
- Use case frameworks that are large scale combinations of use case patterns for an application domain.
- Aspect patterns for wearing non-functional requirements with functional requirements.

Alspaugh et al. [28] have provided an approach to scenario management and evolution. They defined scenarios as a sequence of events with associated attributes. The defined a similarity measure as the sum of the number of common attribute values in each attribute list, divided by the sum of sizes of each attribute list. A variation was also proposed where attributes are assigned weights. Annotations are used to guide matching. The authors have not taken into account the relation between the attributes that might arise from semantic structures.

TABLE I. EARLY SOFTWARE ARTIFACT REUSE EFFORT

Work	Artifact Support	Internal Rep.	Matching Criteria	Extra Annotation	Search Algorithm
[6]	Sequence Diagrams	Message flow	Both Syntactic and Semantic Similarity (Macro and Micro Levels)	No	Depth First B&B Genetic Algorithm
[25] [26]	Use-Cases, Sequence Diagrams, Class Diagrams	Conceptual Graphs	Semantic Similarity or Structural relationship similarity (graph matching)	No	Subdue Algorithm
[2]	Class Diagrams	Class Vectors	Syntactic	No	Greedy & Hungarian Algorithm
[27]	Use Cases	Use Case Templates, Use Case Patterns, Use Case Framework, Aspect Patterns	Analogy-based matching	No	N/A
[28]	Scenarios	Attributes or facets constituting the use case scenario (Actor, go, purpose etc)	Syntactic similarity between scenarios	Yes	N/A
[3]	Use Cases, Sequence diagrams	Event flow vectors	Both Syntactic and Semantic Similarity	Yes	N/A

Bloch and Cybulski [3] have considered event flows in use-cases in measuring the use-case similarity. The authors represented the event flows present in the use-cases as follows. They classified the various events that are part of a particular domain model into a set of clusters. For a new use case they associate its events to these clusters based on the lexical description of the event. The entire event flow in the use case is represented as a vector where each dimension of the vector represents the number of events in a particular cluster.

Ryan and Mathews [29] have facilitated the reuse of previously developed requirement specifications for same or similar domains by identifying and encoding the types of knowledge used during requirement acquisition. They developed a tool (ReqColl) to aid in the process of requirement collection using conceptual graphs with semantic relationships. They used a conceptual graph matching algorithm to compare requirements expressed as conceptual graphs. The similarity is assessed based on matching nodes and arcs inside the conceptual graphs.

Reubenstein and Waters [30] developed a tool, Requirement Apprentice (RA), to assist users in

 ISBN: 978-1-61208-165-6

documenting consistent and complete requirements. The tool maintains a cliché library and uses it to retrieve similar requirements based on hybrid reasoning system. The cliché library holds bulk of the general information related to the domain. This allows users to document only specific requirement information; remaining general information is completed from the library.

In conclusion, even though the UML has become more or less the *de facto* standard modeling language for representing analysis as well as design artifacts, researchers have done little in proposing a similarity metric for UML models. The above discussed works consider comparison of artifacts taking only a single view into consideration. However, as mentioned earlier, a single view cannot comprehensively capture the software requirements and considering only one view while assessing similarity for reuse will not be as effective approach. Accordingly, for more effective reuse of software artifacts the similarity must be assessed considering all views.

Effective IREs should be built on top of strong foundations for a multi-view similarity metric, i.e., a similarity metric that considers multiple views of software in its similarity assessment.

IV. RESEARCH QUESTIONS

The problem of multi-view similarity assessment thus can be stated as to map classes of class diagrams, sequence diagrams, and state machine diagrams of the input model to particular counterparts in the repository such that the substitutions (e.g., class mappings from input model to repository model) have least conflicts. This problem is in essence a constraint satisfaction combinatorial optimization problem in a possibly large, but finite, space [6]. Accordingly, finding optimal substitution for maximal similarity of UML artifacts represents a NP-hard problem. The applicability of search heuristic algorithms, to include but not limited to Genetic Algorithms, Tabu Search, and Simulated Annealing, should be investigated [31][32][33][34]. Performance comparison against exact exhaustive search techniques, e.g., Depth-first Branch and Bound, should also be considered.

Similarity assessments and corresponding measurements should be used in ranking repository projects models according to their similarities to a current project models.

Cornerstone to the similarity assessment is the multi-view similarity metrics. Effective metrics, according to some measures such as *precession* and *recall*, should be developed.

A major focus of the IRE should be to offer tools to facilitate reusing design and later artifacts based on matching requirements. However, the IRE should also offer ad-hoc semantic-based UML artifacts repository search. Metadata and ontology along with indexing and storing schemes to allow for time-wise efficient retrieval of previous artifacts from the repository should also be developed. Standard representation in line with standards such as the *resource description framework schema* (RDFS) should be investigated for repressing the metadata and the ontology [17]. Text/data mining algorithms should be researched and developed to support such search activities.

For maximal utilization within the day-to-day activities, best ways for integration with CASE tools should be researched. The market is glutted with UML modeling tools such as Rational Rose, Enterprise Architect, Together J, Visio, Microsoft Visual Modeler, Advanced Tech GD-Pro, Visual UML, Object Domain, Object Team, etc. The standard interchange format adopted for the UML is the XML Metadata Interchange (XMI) format [37]. Using XMI in representing UML artifacts in the repository should be investigated.

It is also important to maintain the quality of the artifacts maintained in the repository; and in the same line offer recommendations to re-users with regard to such artifacts. In order to do so, the IRE should facilitate the collection of reviews from re-users with respected to re-used/inspected artifacts. Algorithms should be developed to allow synthesizing recommendations with regard to existing artifacts based on a diversity of reviews as well as some other statistics.

Last but not least, intelligent user interface for easy artifacts management and reuse within the context of the day-to-day development activities should be developed. It should be intelligent in the sense that it can offer suggestions with regard to the modeling task at hand.

V. CONCLUSION AND FUTURE WORK

This paper presented an approach towards the development of integrated UML reuse environment. The paper surveys current state of the art and identifies gaps along with corresponding research questions for future work.

The author has been supervising a group of graduate students in an effort that aims at laying the foundations for multi-view similarity metrics and assessment along with the development of IREs proof of concept through a set of case studies. The effort has started earlier [2][6], was suspended for some time, and got resumed recently as a focus for future work. The successful realization of such IREs to facilitate early-stage reuse of UML artifacts is expected to improve the software quality and developers productivity.

ACKNOWLEDGMENT

The author wish to acknowledge King Fahd University of Petroleum and Minerals (KFUPM) for the use of various facilities in carrying out this research.

The author also thanks Jarallah AlGhamdi, Raheem Rufai, Awes Ahmed, and Sajjad Mahmoud for many helpful discussions, comments, and different contributions to the origin of this work.

REFERENCES

[1] I. Sommerville, *Software Engineering*, 9th ed.: Addison-Wesley, 2010.

[2] R. Rufai, "New Structural Similarity Metrics for the UML," MS Thesis, King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia, 2003.

[3] M. C. Blok and J. L. Cybulski, "Reusing UML Specifications in a Constrained Application Domain," in *Proceedings 5th Asia Pacific Software Engineering Conference (ASPEC'98)*, 1998, pp. 196-202.

[4] V. De Mey and O. Nierstrasz, "The ITHACA Application Development Environment. Visual Objects (ed. D. Tsichritzis," Centre Universitaire d'Informatique, University of Geneva1993.

[5] D. C. Rine, "Success factors for software reuse that are applicable across domains and businesses. ," in *Proceedings of the ACM Symposium on Applied Computing*, 1997, pp. 182-186.

[6] A. Ahmed, "Functional Similarity Metric for UML Models," MS Thesis King Fahd University of Petroleum & Minerals, 2006.

[7] V. Sugumaran, *et al.*, "Software product line engineering," *Communications of the ACM*, vol. 49, pp. 29-32, 2006.

[8] G. Booch, *et al.*, Object-Oriented Analysis and Design with Applications: Addison-Wesley, 2007.

[9] R. M. M. Braga, *et al.*, "Odyssey: A Reuse Environment Based on Domain Models," in Proceedings 1999 IEEE Symposium on Application-Specific Systems and Software Engineering and Technology, ASSET '99., 1999.

[10] J. L. Cybulski, "Introduction to Software Reuse," Technical Report TR96/4, University of Melbourne, Melbourne, Australia1996.

[11] J. L. Cybulski and K. Reed, "Requirements Classification and Reuse: Crossing Domain Boundaries," in *Proceedings of the 6th International Conerence on Software Reuse: Advances in Software Reusability*, London, UK, 2000.

[12] R. K. Keller, *et al.*, "Design and Implementation of a UML-Based Design Repository. ," in *Proceedings 13th International Conference on Advanced Information Systems Engineering (CAiSE2001)*, Interlaken, Switzerland, 2001.

[13] H.-Y. Lee and M. T. Harandi, "An Analogy-Based Retrieval Mechanism for Software Design Reuse.," in *Proceedings of the 8th Knowledge-Based Software Engineering Conference (KBSE'93)*, Chicago, 1993, pp. 152-159.

[14] O. Edelstein, *et al.*, "eColabra: An Enterprise Collaboration & Reuse Environment," in *Proc. Fourth Int'l Workshop (NGITS '99)*, 1999, pp. 229-236.

[15] H. Mili, *et al.*, "Reusing Software: Issues and Research Directions," *IEEE Transactions on Software Engineering*, vol. 21, 1995.

[16] I. Jacobson, *et al.*, Object-oriented software engineering: a use case driven approach: Addison-Wesley, 1992.

[17] W3C, "RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-schema-20040210 last accessed Jan. 2011.," ed, 2004.

[18] A. L. Correa, *et al.*, "Object Oriented Design Expertise Reuse: An Approach Based on Heuristics, Design Patterns and Anti-Patterns," in *6th International Conference on Software Reuse*, 2000, pp. 33 - 191.

[19] H.-J. Beyer, *et al.*, "Introducing Architecture-Centric Reuse into a Small Development Organization," in *Proceedings of the 10th international conference on Software Reuse: High Confidence Software Reuse in Large Systems* Berlin, Heidelberg, 2008.

[20] A. C. Martins, *et al.*, "Suggesting Software Components for Reuse in Search Engines Using Discovered Knowledge Techniques," in *SEAA '09 Proceedings of the 2009 35th Euromicro Conference on Software Engineering and Advanced Applications* 2009, pp. 412-419.

[21] J. Rumbaugh, *et al.*, *Object-Oriented Modeling and Design*, First ed.: Prentice Hall, 1991.

[22] J. Iivari, "Object-orientation as structural, functional and behavioural modelling: a comparison of six methods for object-oriented analysis," *Information and Software Technology*, vol. 37, pp. 155-163, 1995.

[23] G. Kotonya and I. Sommerville, *Requirements Engineering: Processes and Techniques*: John Wiley & Sons, 2000.

[24] J. Mylopoulos, "Multiple Viewpoints Analysis of Software Specification Process," *IEEE Transactions on Software Engineering*, 1995.

[25] W. N. Robinson and H. G. Woo, "Finding Reusable UML Sequence Diagrams Automatically," *IEEE Software*, vol. 21, pp. 60-67, 2004.

[26] H. G. Woo and W. N. Robinson, "Reuse of Specifications Using an Automated Relational Learner: A Lightweight Approach," in *Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering*, 2002, pp. 173-180.

[27] M. Saeki, "Patterns and Aspects for Use Cases: Re-use Techniques for Use Case Descriptions," in *Proceedings of the 4th International Conference on Requirements Engineering (ICRE'00)*, 2000.

[28] T. A. Alspaugh, *et al.*, "An Integrated Scenario Management Strategy," in Proceedings of the 4th IEEE International Symposium on Requirements Engineering, 1999.

[29] K. Ryan and B. Mathews, "Matching Conceptual Graphs as an aid to Requirements Re-use," *Proceedings of IEEE International Symposium on Requirements Engineering*, pp. 112-120 1993.

[30] H. B. Reubenstein and R. C. Waters, "The Requirements Apprentice: Automated Assistance for Requirements Acquisition," *IEEE Transactions on Software Engineering*, vol. 17, 1991.

[31] R. Poli, *et al.*, "Particle swarm optimization: An overview," *Swarm Intelligence*, vol. 1, pp. 33-57, 2007.

[32] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed.: Prentice Hall, 2010.

[33] S. Sait and H. Youssef, Iterative Computer Algorithms with Applications in Engineering: IEEE Computer Society, 1999.

[34] D. Teodorovic, *et al.*, "Bee Colony Optimization: Principles and Applications," in 8th Seminar on Neural Network Applications in Electrical Eng., NEUREL-2006, IEEE CNF, 2007.

[35] L. Chung and K. Cooper, "Knowledge based COTS aware requirements engineering approach," in 14th Int. Conf. Software Eng. Knowledge Eng, 2002, pp. 175 - 182.

[36] D. J. Carney, *et al.*, "Identifying Commercial off the Shelf Product Risks: The COTS Usage Risk Evaluation," Carnegie Mellon Software Engineering Institute (SEI) 2003.

[37] Object Management Group (OMG). OMG XML Metadata Interchange (XMI) Specification (V. 2.11/Beta 2.4). Jan. 2002. http://www.omg.org/spec/XMI/ (Accessed 2011-05-28).

[38] B. Sheiderman and C. Plaisant, Designing the User Interface, Pearson Education, Inc., 2005.

[39] Object Management Group (OMG). OMG Unified Modeling Language (OMG UML), Superstructure (V. 2.3). May, 2010. http://www.omg.org/spec/UML/2.3/Superstructure/PDF/ (Accessed 2011-09-23).

 ISBN: 978-1-61208-165-6