Using SSUCD to Develop Consistent Use Case Models: An Industrial Case Study

Mohamed El-Attar
Information and Computer Science Department
King Fahd University of Petroleum and Minerals
P.O. 5066, Al Dhahran 31261, Kingdom of Saudi Arabia
melattar@kfupm.edu.sa

***Abstract*- In software development projects that utilize a use case-driven development methodology, it is crucial to develop high quality use case models to ensure the development of a quality end product. There are many quality attributes for use case models. One of these qualities is consistency. A structure named SSUCD (Simple Structured Use Case Descriptions) was developed to guide use case authors while authoring their use cases. SSUCD was developed in previous work to specifically tackle the issue of consistency in use case models. In particular, SSUCD ensures structural consistency in use case models. Thus far, SSUCD has been validated using exemplars. While exemplars provide beneficial preliminary validation, a more thorough validation process is required to ensure the industrial applicability of SSUCD. To this end this paper presents an industrial case study that was used to validate SSUCD. The result of the case study shows that SSUCD can be effectively used to develop consistent use case models of industrial strength.**

Keywords – Use Cases; SSUCD; Model Consistency.

I. INTRODUCTION

Use case modeling [7, 13] is a very popular technique used to elicit and model functional requirements in object-oriented software development projects. In a use case driven development methodology, the use case model is used to drive the development of other UML (Unified Modeling Language) [13] design artifacts at the design phase. This process is vulnerable to human injected defects since naturally there is a gap between the analysis and design phases. Consequently, this will cause system architects to create designs that provide different functionality from that was required (i.e., developing the 'wrong' system), leading to costly reworks and schedule overruns, in addition to the intangible cost of unsatisfied customers. It is, therefore essential to develop high quality use case models in order to ensure the development of end systems that delivers the required functionalities; while allowing them to be understandable by all stakeholders, including "non-technical" stakeholders.

The literature has identified a number of use case models quality attributes that can be categorized into five main categories: consistency, correctness, completeness, analytical and understandability [11]. The harmful consequences of lacking in any of these quality attributes have been documented in the literature. Many research works have been devoted towards improving these quality attributes or at least targeting a subset of these quality attributes. Consistency in particular is a highly sought after quality attribute [1-6, 10-12]. The current state of practice to develop use case models depends on the modeler's discipline to create consistent use case models. Such discipline seldom in exists in practice. In previous work, a structure named SSUCD [11] was developed to specifically target the issue of inconsistencies in use case models. In particular, SSUCD can be used to ensure structural consistency in use case models. SSUCD does not directly improve other quality attributes. Therefore, it is recommended that SSUCD be used in addition to other researched techniques to improve the overall quality of use case models.

The remainder of this paper is organized as follows: Section 2 provides a brief background and discusses related works. Section 3 presents the SSUCD structure. In Section 4, the MAPSTEDI case study is presented. Finally, Section 5 concludes and provides suggestions for future work.

II. BACKGROUND AND RELATED WORK

A use case model consists of a use case diagram, a set of use case descriptions and a glossary. The glossary is an artifact that is shared by all artifacts developed in a project to document relative terminology in a consistent manner. The use case diagram serves as a visual summary of the functional requirements of the underlying system. The functional requirements are textually detailed in use case descriptions.

In a use case model, inconsistency can occur between the use case descriptions, the various diagrams (if more than one was used), and most commonly inconsistency may occur between the use case diagrams and the corresponding set of use case descriptions. The cost of inconsistencies depends on the form it exists in.

The literature has repeatedly warned against inconsistencies in use case models. A taxonomy of use case modeling defects and their harmful consequences were presented in Anda et al. [3]. The taxonomy states that inconsistencies in a use case model have a detrimental effect on every aspect of the development process and in turn severely hampering the overall quality of the end product. In Lilly [10], a number of inconsistency defects were outlined. For example, an inconsistent system boundary has been found to cause ambiguity with respect to the functionality that needs to be developed. Development teams may suffer from costly redundant and unnecessary development leading to schedule overruns. Conversely, development teams may miss some of the required functionality. Inconsistencies in use case models has also been found to be symptomatic of an ambiguous domain model and a use case model that might be handling concepts that are not defined or understood properly [5]. Inconsistencies may also be a result of missing or vague information [5]. Ambler [2] warns that a high level of

 ISBN: 978-1-61208-230-1

inconsistencies in use case models may render it useless as it becomes too outdated.

Naturally many research works have been devoted towards improving consistency in use case models. For example, Armour and Miller [6] and Kulak and Guiney [8] have highly recommended various mechanisms of reviewing use case models as means to ensure their quality by assuring that they possess a great deal of consistency. An automated approach was proposed by McCoy [12]. McCoy [12] presents a tool that provides a template for use case authors to write their use cases. The template aids in ensuring consistency during the data entry process. Butler et al. [4] introduced the concept of refactoring to the use case modeling domain. A number of use case refactorings improve consistency.

III. THE SSUCD STRUCTURE

The structure SSUCD was devised to specifically tackle the issue of inconsistencies. SSUCD employs a template of commonly used fields in popular use case description templates such as those presented by Cockburn [1]. Use cases described using the SSUCD structure contains four main sections, these are: (a) Use Case Name, (b) Associated Actors, (c) Description, (d) Extension Points and Extended Use Cases. With the exception of the "Description" section, these sections utilize a handful of keywords to embed the required structure. All keywords are written in uppercase for readability purposes. The "Description" section on the other hand is populated using natural language to allow for maximum flexibility and expressiveness by use case authors. Other sections can be added to cater to specific needs; the additional sections must be contained as subsections of the "Description" section.

The design of the SSUCD structure accounted for readability. This is achieved by using a limited set of English keywords that are inserted within various sections of the templates. All keywords pertain to the use case modeling domain and thus greatly reducing the required learning curve. A brief description of each keyword is shown in Table 1. Figures 1 and 2 illustrates the concepts explained above and demonstrates the visually the mapping of the keywords in Table 1 using a mock example.

Table 1 A summary of the SMCD structure constructs

Section	Keyword	Diagram Representation
Use Case Name	ABSTRACT	*Abstract* use cases are depicted in italic font in the diagrams.
	SPECIALIZES	A generalization relationship link is depicted in the diagram.
	IMPLEMENTS	A generalization relationship link is depicted in the diagram. This is due to the fact that the generalization and implementation relationships are depicted using the same notation.
	The name of the use case	A use case with the given name is displayed in the diagram.
Description	INCLUDE	Results in the creation of an *include* relationship directed towards the use case stated in the INCLUDE statement.
Extended Use Cases	Base Use Case	An *extend* relationship link is created and directed towards the stated base use case.
	Extension Point	Optional to the user. Results in the augmentation of the targeted extension point name on the *extend* relationship link.
	IF	Optional to the user. The condition is displayed on the *extend* relationship link in square brackets.
Extension Points	The names of public extension points	Each extension point stated is depicted within the oval of the given use case in the diagram.

Mock Example Textual Descriptions
Actor Name: A **Brief Description:** A brief description of actor A
Actor Name: B **SPECIALIZES:** A **Brief Description:** A brief description of actor B
Use Case Name: C **ABSTRACT** **Brief Description:** A brief description of use case C
Use Case Name: D **Brief Description:** A brief description of use case D **Extended Use Cases:** **Base UC Name:** F **AT**: extension point of F **IF**: is true
Use Case Name: E **IMPLEMENTS:** C

 ISBN: 978-1-61208-230-1

SPECIALIZES: D

Brief Description:
A brief description of use case E and INCLUDE <F>

Use Case Name: F

Brief Description:
A brief description of use case F.

Extension Points:
Extension point of F

Figure 1. Mock Example of Textual Descriptions

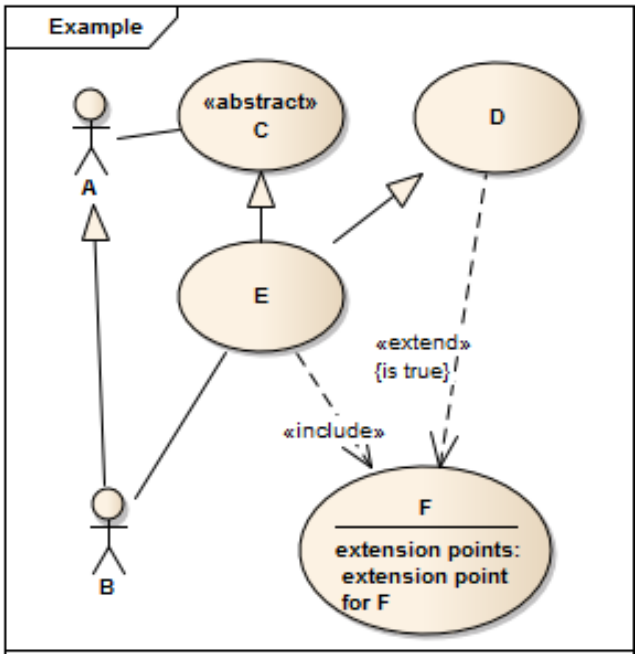

Figure 2. Example use case diagram including the entire notational set supported by the SSUCD structure

The SSUCD structure is supplemented with the REUCD (Reverse Engineering of Use Case Descriptions) process. There are two perform key functionalities that are performed by REUCD [11]: (a) REUCD constructs a use case diagram that accurately represents the textual descriptions of use cases and actors, (b) REUCD can generate skeletons of use case descriptions. Once these descriptions are completed, REUCD can once again be used to generate a use case diagrams that accurately represents the textual descriptions.

A. Consistency and Mapping Rules Between Use Case Descriptions and Diagrams

In this section, we will introduce the REUCD (Reverse Engineering of Use Case Diagrams) process, which is used to systematically map SSUCD's structural constructs to diagrammatic notations that form use case diagrams. This systematic process is automated using the tool SAREUCD (see Section 5), which will ensure the consistency and speed of the process.

The process of generating use case diagrams from use case descriptions and vice versa is analogous to generating complete and accurate UML class diagrams from code and generating code structures from UML class diagrams. The reason UML class diagrams cannot be used to generate complete programs is because they act as a visual summary of a program's static structure. UML class diagrams are at a higher level of abstraction compared to code. On the other hand, a complete program will contain more than enough details required to generate complete and accurate UML class diagrams.

Use case descriptions (analogous to code) contain far more details than use case diagrams (analogous to class diagrams). Use case diagrams are at a higher level of abstraction than the descriptions. Therefore, given a set of use case descriptions, a complete and accurate use case diagram can be systematically produced. However, if modelers choose to create use case diagrams manually first, which is often the case; a 'skeleton' of the use case descriptions can be systematically produced. Detailed descriptions of the use case are later added manually by analysts to 'flesh out' the generated 'skeletons'. After the use case descriptions are complete, an updated version of the use case diagram can be systematically generated. Users of SSUCD and REUCD will not be burdened with performing these transformations since they will be carried out by a tool.

B. The REUCD Process

When given a set of SSUCD use case description, the REUCD process is applied by iteratively parsing through the text of the descriptions. Each iteration has several purposes and these are described below:

- **Iteration 1**: Identify actors and create XML components to represent these actors to be displayed by a UML modeling tool.
- **Iteration 2**: Identify use cases and create XML components to represent these use cases to be displayed by a UML modeling tool.
- **Iteration 3**: Identify relationships between actors and use cases and create to corresponding XML components. This step will require cross-referencing with XML components previously created in the previous two iterations.

When given a use case diagram, the REUCD process is applied on the XML file the represents the given use case diagram. The process is applied by iteratively parsing through the text of the XML file. Each iteration has a purpose as defined below:

- **Iteration 1:** Identify actors and create a text area for each actor with its name and the appropriate fields.
- **Iteration 2:** Identify use cases and create a text area for each use case with its name and the appropriate fields.
- **Iteration 3:** Identify the relationships between actors and use cases and amend the corresponding text area to reflect these relationships.

Finally, the text areas are combined into one file.

IV. THE MAPSTEDI SYSTEM CASE STUDY

In this section, we present an industrial case study where SSUCD was applied successfully. This case study is

 ISBN: 978-1-61208-230-1

concerned with the MAPSTEDI (Mountains and Plains Spatio-Temporal Database Informatics) use case model [9]. The MAPSTEDI system was built for research purposes by geocoders to help them analyze biodiversity data in the northern plains as well as the southern and central Rocky Mountains both spatially and temporally. It was developed by the Denver Botanic Gardens (DBG), Denver Museum of Nature and Science (DMNS) and University of Colorado Museum (UCM). The project's aim is to merge their separate collections into one distributed biodiversity database to include over 285,000 biological specimens.

The use case model of the MAPSTEDI system originally five use case models representing five subsystems. The use case diagrams of three subsystems were later merged as a result of a refactoring process. A brief description of each subsystem is provided below:

- **Database Queries:** The purpose of this subsystem is to perform queries on local and distributed databases for collections data. There are two distributed databases.
- **Database Integrator:** The purpose of this subsystem is to handle how the collections data from separate databases are integrated after being updated.
- **Database Edits:** The purpose of this subsystem handles the operational mechanisms for editing and updating the databases. The databases are updated whenever a geocoder edits the collections data.
- **Administrative Process:** The purpose of this subsystem outlines the administrative functionalities and responsibilities. This subsystem backups and restores collections data and application code. Moreover, the subsystem is used to install any new updates.
- **Database Access:** The purpose of this subsystem handles access control of the database; who may access the database and how. Public users have access to search and download collections data and visualize biodiversity analysis. However, only researchers have access to sensitive data.

The "Database Access" and "Administrative Process" subsystems each had a separate use case diagram. Meanwhile, the "Database Edits", "Database Queries" and "Database Integrator" subsystems are represented by a single merged use case diagram.

The purpose of this case study is to validate the SSUCD structure and the REUCD process. In this case study, the use case and actors descriptions were developed using the SSUCD structure. The textual descriptions were then used as input by the REUCD process to produce the corresponding use case diagrams. The successful application of this case study is if use case diagrams generated by the REUCD process were structurally similar. Figures 1, 3 and 5 below contain the textual descriptions of the use cases and actors in each use case diagram. The use case diagrams generated by REUCD based on the descriptions in Figure 1, 3 and 5, are shown in Figure 2, 4, and 6, respectively.

Database Access

Actor Name:
User

Brief Description:
<A brief description about the User actor>

Actor Name:
Public User

Specializes:
User

Brief Description:
<A brief description about the Public User actor>

Actor Name:
Research User

Specializes:
User

Brief Description:
<A brief description about the Research User actor>

Use Case Name:
Download Collections Data

Associated Actors:
User

Basic Flow:
… INCLUDE <Search Collections Data>

Use Case Name:
Search Collections Data

Associated Actors:
User

Basic Flow:
…this use case allows the user to search collections data…

Use Case Name:
Visualize Biodiversity Analysis

Associated Actors:
User

Basic Flow:
…this use case allows the user to visualize biodiversity analysis…

Use Case Name:
Access Sensitive Data

Associated Actors:
Research User

Basic Flow:
…this use case allows the research user to access sensitive data…

Figure 3. The descriptions of the use cases and actors of the "Database Access" subsystem

 ISBN: 978-1-61208-230-1

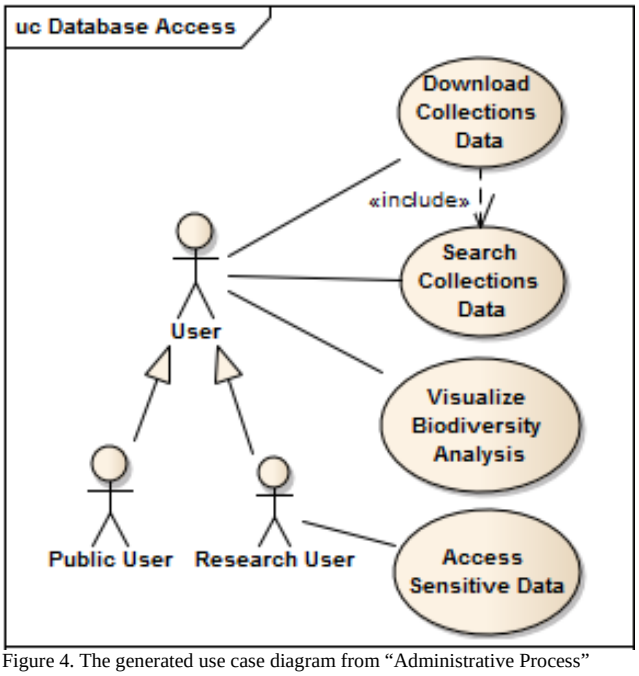

Figure 4. The generated use case diagram from "Administrative Process"

Administrative Process
Actor Name: Administrator **Brief Description:** <A brief description about the Administrator actor>
Actor Name: Database Administrator **Specializes:** Administrator **Brief Description:** <A brief description about the Database Administrator actor>
Actor Name: ArcIMS Administrator **Specializes:** Administrator **Brief Description:** <A brief description about the ArcIMS Administrator actor>
Use Case Name: Backup Process **Associated Actors:** Administrator **Basic Flow:** …this use case name allows the administrator to perform process backup…
Use Case Name: Restore Process **Associated Actors:** Administrator **Basic Flow:** …this use case name allows the administrator to perform process restoration…
Use Case Name: Install Software Updates **Associated Actors:** Administrator **Basic Flow:** …this use case name allows the administrator to install software updates…

Figure 5. The descriptions of the use cases and actors of the "Administrative Process" subsystem

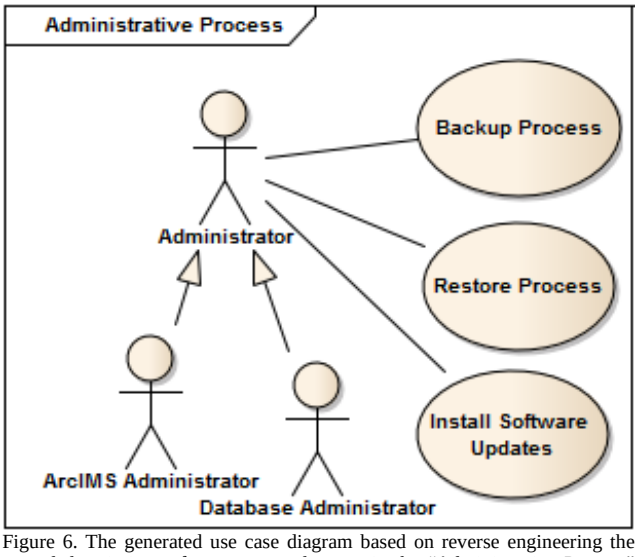

Figure 6. The generated use case diagram based on reverse engineering the textual descriptions of use cases and actors in the "Administrative Process" subsystem.

Merged Subsystems
Actor Name: Geocoder **Brief Description:** <A brief description about the Geocoder actor>
Actor Name: Database Integrator **Brief Description:** <A brief description about the Database Integrator actor>

 ISBN: 978-1-61208-230-1

Use Case Name:
Geocode Specimen

Associated Actors:
Geocoder

Basic Flow:
…this use case name allows the administrator to geocode a specimen and it INCLUDE <Update Collections Data>…

Use Case Name:
Update Collections Data

Associated Actors:
Database Integrator

Basic Flow:
…this use case name allows the administrator to update collections data…

Extended Use Cases:
Base Use Case Name: Query Remote Database

Use Case Name:
Query Remote Database

Specializes:
Query Database

Basic Flow:
…this use case name allows the administrator to query remote database…

Use Case Name:
Query DMNS Database

Specializes:
Query Remote Database

Basic Flow:
…this use case name allows the administrator to query DMNS database…

Use Case Name:
Query DIGIR Database

Specializes:
Query Remote Database

Basic Flow:
…this use case name allows the administrator to query DIGIR database…

Use Case Name:
Query Database

Basic Flow:
…this use case name allows the administrator to query database…

Use Case Name:
Query Local Database

Specializes:
Query Database

Basic Flow:
…this use case name allows the administrator to query local database…

Use Case Name:
Integrate Query Results

Associated Actors:
Database Integrator

Basic Flow:
…this use case name allows the administrator to integrate query results and it INCLUDE <Query Remote Database> and INCLUDE <Query Local Database>…

Figure 7. The descriptions of the use cases and actors of the merged subsystems

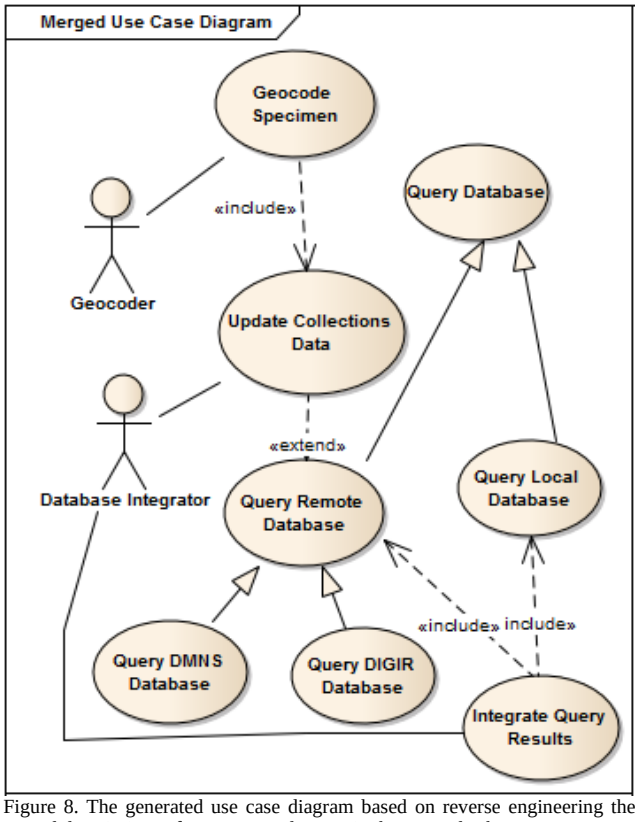

Figure 8. The generated use case diagram based on reverse engineering the textual descriptions of use cases and actors in the merged subsystems.

A. Verifying the Correctness of the Generated Use Case Diagrams

The correctness of the generated use case diagrams was verified through two distinct means. The first approach involved the use of the *UseCaseDiff* tool [14] to check for differences between the generated use case diagrams and the original use case diagrams. UseCaseDiff is an open source use

 ISBN: 978-1-61208-230-1

case diagram differencing tool that was developed as part of previous work [14]. Both sets of use case diagrams were provided as input into the *UseCaseDiff* tool. The tool generated a report showing no structural differences.

The second approach used to verify the correctness of the generated use case descriptions was via manual inspection. The two sets of diagrams were juxtaposed manually by three independent researchers. The reviewers did not find any structural differences between the two sets of diagrams.

V. Conclusion and Future Work

In this paper, we report on the successful use of SSUCD to develop a structurally consistent industrial use case model that represents the functionality of the five subsystems comprising the MAPSTEDI system. The case study has shown that SSUCD can be utilized by industry practitioners to develop consistent use case models and to help them detect structural inconsistencies in existing models.

Future work can be directed towards developing an approach to transform use cases written using SSUCD into other types of models, such as UML Activity and Sequence Diagrams.

Acknowledgements

The author would like to acknowledge the support provided by the Deanship of Scientific Research (DSR) at King Fahd University of Petroleum & Minerals (KFUPM) for funding this work.

References

[1] A. Cockburn, Writing Effective Use Cases. Addison-Wesley, 2000.

[2] S. Ambler, Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process. Wiley, 2002.

[3] B. Anda, D. Sjøberg, and M. Jørgensen, "Quality and Understandability in Use Case Models," 15th European Conference Object-Oriented Programming (ECOOP), edited by J. Lindskov Knudsen. Springer-Verlag, Budapest, Hungary, pp. 402-428, 2001.

[4] G. Butler and L. Xu, "Cascaded refactoring for framework evolution," Proceedings of 2001 Symposium on Software Reusability, ACM Press, pp. 51-57, 2001.

[5] P. Chandrasekaran, "How Use Case Modeling Policies Have Affected The Success of Various Projects (or How to Improve Use Case Modeling)," Addendum To The 1997 ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, pp. 6-9, 1997.

[6] F. Armour and G. Miller, Advanced Use Case Modeling. Addison-Wesley, 2000.

[7] I. Jacobson, M. Ericsson, and A. Jacobson, The Object Advantage. ACM Press, 1995.

[8] D. Kulak and E. Guiney, Use Cases: Requirements in Context. Addison-Wesley, 2000.

[9] M. El-Attar, Analysis of the MAPSTEDI system Use Case Model. Available online: http://www.steam.ualberta.ca/main/research_areas/MAPSTEDI%20Analysis.htm. [retrieved: October 2012].

[10] S. Lilly, "Use Case Pitfalls: Top 10 Problems from Real Projects Using Use Cases," Proceedings of TOOLS USA '99, IEEE Computer Society,- pp. 174-183, 1999.

[11] M. El-Attar and J. Miller, "Producing Robust Use Case Diagrams via Reverse Engineering of Use Case Descriptions," Journal of Software and Systems Modeling, vol. 7, no. 1, pp. 67-83, 2008.

[12] J. McCoy, "Requirements Use Case Tool (RUT)," Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, pp. 104-105, 2003.

[13] OMG 2003, "UML Superstructure Specification", Object Management Group, http://www.omg.org/docs/ptc/03-08-02.pdf, 2003. [retrieved: October 2012].

[14] M. El-Attar, "UseCaseDiff: An Algorithm for Differencing Use Case Models," 9th International Conference on Software Engineering Research, Management and Applications, pp. 148-152, 2011.

 ISBN: 978-1-61208-230-1