International Journal of Advanced Smart Convergence Vol.10 No.2 130-137 (2021)
http://dx.doi.org/10.7236/IJASC.2021.10.2.130

IJASC 21-2-17

Quality Improvement by enhancing Informal Requirements with Design Thinking Methods

Janghwan Kim[1], R. Young Chul Kim[2*]

[1]M.S, Software Engineering Laboratory, Department of Software and Communication Engineering, Hongik University, Korea
[2]Professor, Software Engineering Laboratory, Department of Software and Communication Engineering, Hongik University, Korea
[1]janghwan@selab.hongik.ac.kr, [2]bob@hongik.ac.kr

Abstract

In the current software project, it is still very difficult to extract and define clear requirements in the requirement engineering. Informal requirements documents based on natural language can be interpreted in different meanings depending on the degree of understanding or maturity level of the requirements analyst. Also, Furthermore, as the project progresses, requirements continue to change from the customer. This change in requirements is a catastrophic failure from a management perspective in software projects. In the situation of frequent requirements changes, a current issue of requirements engineering area is how to make clear requirements with unclear and ambigous requirements. To solve this problem, we propose to extract and redefine clear requirements by incorporating Design Thinking methodologies into requirements engineering. We expect to have higher possibilities to improve software quality by redefining requirements that are ambiously and unclearly defined.

Keywords: *Software Engineering, Software Development Life Cycle, Design Thinking, Requirement Engineering, NLR*

1. Introduction

The whole world is now facing an unprecedented major transformation since the Industrial Revolution. With the development of computer technology and Internet technology, the size of the industry has grown and diversified, and many studies have been conducted to produce high-level software quickly and accurately. As software quality advancement is required along with a variety of software types, analysis, and design of requirements in software development projects are critical factors to make or break software projects [1]. Because software development processes analyze requirements, design software based on analyzed information, and implement based on design, each step of the process is deeply related to each other. Requirement analysis is the task of specifying and organizing what type of software will be created from customers and how it functions and has a huge impact on the entire software development process, such as design, development, and testing after analysis. However, because customers are not usually developers, they

Manuscript Received: May. 9, 2021 / Revised: May. 13, 2021 / Accepted: May. 16, 2021
Corresponding Author: bob@selab.hongik.ac.kr
Tel: +82-44-860-2477
Professor, Software Engineering Laboratory, Department of Software and Communication Engineering, Hongik University, Korea

often describe the software from the perspective of the average person rather than speaking in the language of the developer. In many cases, the meaning of the requirements is unclear. This enables requirement analysts and those participating in the development to interpret different meanings rather than those meanings of the customer's requirements [2]. In addition, when a requirement analyst analyzes the requirements in the natural language of the ordering entity, the analysis of the requirements may vary depending on the analyst's experience and abilities. To address these issues, we propose a way to apply design-thinking mechanisms to better analyze natural language requirements where the meaning of the requirements is inaccurate by applying them to software development processes. In Gary E. Mogyorodi's paper, removing ambiguity of the requirements clarifies the specification of the requirement, and it soon appears to improve the quality of the requirement [3].

To address these issues, we propose a method to apply design-thinking mechanisms to software development processes to clearly analyze requirements in natural language requirements. This paper is a research to simplify the requirement ambiguity using design-thinking mechanisms to enhance the clarity of the requirements and thereby improve the quality of the requirements. We expect to improve the accuracy of the design based on the improved requirements through this method and also improve the quality of the software.

This paper is organized in the following order. Section 2 refers to software development life cycles and the requirement engineering as related studies and introduces design thinking mechanisms. Section 3 introduces the definition of an unclear requirement and refers to how to improve the requirement specifications. Section 4 applies the proposed method through case studies, and section 5 remarks future research directions with conclusions.

2. Related Studies

2.1 Design Thinking

Design Thinking(DT) is a "thinking process" for designers to produce end-products [4]. DT helps creative thinking through the following processes. It includes steps as follows. Empathize - Define - Idea - Prototype - Test. There are many methodologies to carry out the process, such as AEIOU, 5W1H, etc. Moreover, design thinking is a very useful technique for solving problems by redefining unclear and unknown problems [5]. Each step in DT does not always have to be sequential, it can occur in parallel, and each step can be repeatedly performed [4]. That is, each step should not be understood as a hierarchical or step-by-step process.

2.2 Software development life cycle and requirements engineering

The software development life cycle is a term that refers to the software development process applying engineering techniques in a software project and represents the overall process of software development. The software development life cycle consists of requirements analysis - design - implementation - verification - operation [6].

Requirements engineering is an important factor in the success of a software project, and the larger the size of the software project, the greater the effect [1]. Requirements engineering refers to the process of defining requirements by applying the engineering design process and managing the requirements by documenting them. Requirements engineering consists of following steps: requirements elicitation - requirement analysis - system modeling - requirements specification - verification - management.

The requirements elicitation stage refers to the process of inquiring about the requirements for the software product to be developed by meeting the stakeholders of the project, customers, and the developer [1]. In this stage, requirements analysts usually extract requirements based on product user behavior [7]. In the requirements analysis step, the client, developer, and stakeholders identify and define requirements based on the elicited requirements. The requirements analysts then document the defined requirements and model the system based on them. In the requirements verification stage, based on the documented content, the requirements specification is completed by verifying that all requirements are consistent and satisfy the requirements of stakeholders.

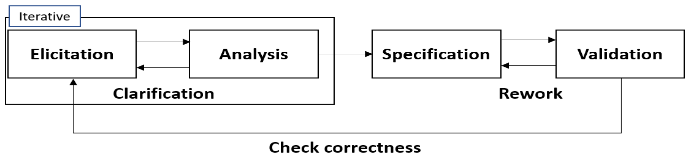


Figure 1. Requirement Engineering Process

3. Enhancing the quality of unclear requirement specifications

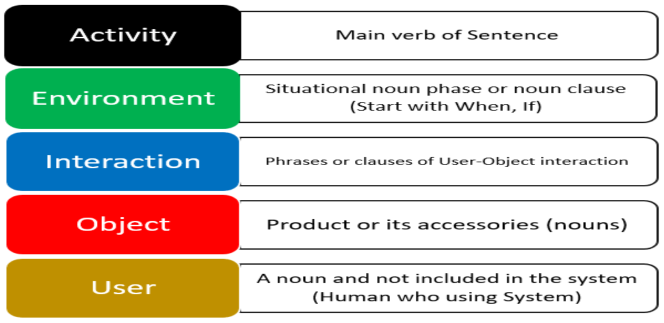
A software project starts with figuring out what the customer wants. In order to introduce a way to improve an unclear requirement, a definition of what an unclear requirement should be preceded. Definitions of these requirements are as follows:

B. Davey remarks on his paper "Unclear requirements arise in various forms with various reasons such as, Incomplete requirements, Incomplete domain knowledge, Overlooking tacit assumptions, Ill-defined system boundaries, Ambiguous requirements, Un-testable terms, etc." [8].

In our paper, we focus on improving the problem of ill-defined system boundaries and ambiguous requirements. When the requirements of customers and various stakeholders are extracted, requirements analysts and developers begin to analyze the requirements together. The proposed steps is as follows:

Step 1. Requirement Analysis by Breaking Requirement Sentence by Sentence.

First, we divide the Natural Language Requirements (NLRs) into sentence units. And then, Analyzing the requirements of each sentence unit through the empathy technique of Design Thinking(DT). Those requirements are classified into five different categories by applying the AEIOU technique, which is often used in the empathy stage of DT. Figure 2 shows a rule of classification of a sentence unit. 'A' is an abbreviation for Activity and is classified as the main verb of the sentence. 'E' is an abbreviation for Environment and represents the 'environment' in which the requirements are performed. The environment mainly represents the conditions in which the function of the requirement (mainly the Activity) occurs. 'I' is an abbreviation for Interactions and represents the part where User and Object interact. 'O' is an abbreviation of Objects and represents all noun objects except User in the requirements. Object mainly represents the product to be made. 'U' is an abbreviation for Users, and it represents the user of the software or the subject of interaction with the object in the requirements. Usually, the subject of a sentence is classified as user. As



Activity	Main verb of Sentence
Environment	Situational noun phase or noun clause (Start with When, If)
Interaction	Phrases or clauses of User-Object interaction
Object	Product or its accessories (nouns)
User	A noun and not included in the system (Human who using System)

Figure 2. AEIOU mapping rule for classification

shown in Figure 2 above, the sentences divided according to each category are classified and mapped in the natural language requirements using the 'AEIOU technique'

Step 2. One sentence has one meaning by using 5W1H

When the requirements are processed through the AEIOU technique in Step 1, the extracted requirements information is put back together to redefine the sentence. The sentences redefined through this method become semantically clearer than natural language sentences [10]. The word 'clear' in a sentence means that "one sentence should convey only one meaning, avoiding expressions that can be interpreted in many ways" [11]. Moreover, this redefined requirement was a result of elimination of requirement factors that doesn't affect the requirement specification. Therefore, the requirements that are redefined through the proposed rule are more semantically clear than original requirements when they are listed in natural language form.

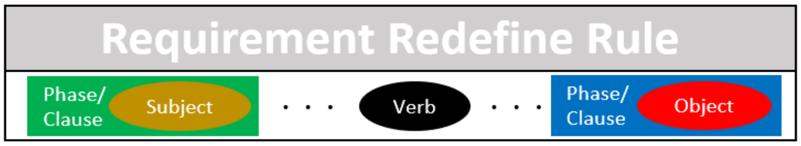


Figure 3. Requirement Classification

Step 3. System design from enhanced requirements specification

In step 3, requirements modeling is performed using use case technique, which is one of the requirements engineering management techniques. Use-case diagram is well-known high-level system design techniques. Actors in Use-case diagram are human being that use or participate the case scenarios in the diagram[12].

In this step, we represent Use-case Diagram based on the Actor, Object, and Interaction information that are extracted from the previous step. This diagram combines Activity and Object to create Use-case scenario, and the user that is interacting with Scenario is the Actor, and the Interaction is mapped with arrows. Use-case Diagram intuitively demonstrates the functionality of the software and indicates the interaction of the software with the user [13]. Such methods represent the requirements as Use-case Diagram based on element-by-element information and design from the requirements redefined by 5W1H, thus improving the clarity of the design over those designed directly from the requirements without mapping with the analysis method proposed in this paper. Figure 4 shows how extracted elements are mapping on use-diagram. An actor represents a user and activities represent main verbs of the requirements. Lastly, an arrow between activity and actor represents the interaction.

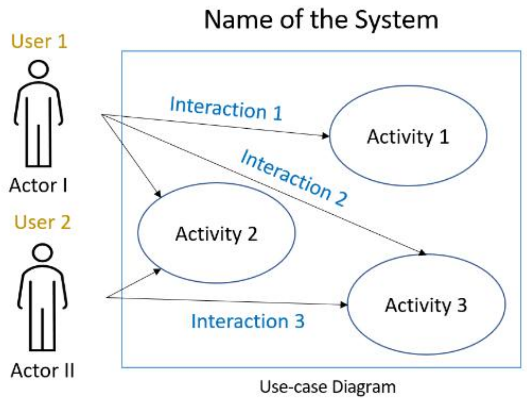


Figure 4. Use-Case Diagram for System Design

Step 4. Abstract Design class with sequence diagram

Sequence diagrams show how functions are performed based on 'actor' sequentially. The sequence diagrams not only show in what order the functions perform, but also show the flow of data when performing the functions. As we show on figure 5, it contains actors, and objects. Those arrow shows messages or data flow between actor and object or an object and another objects. Even though it is not necessary to have return value, it has enough to say that those objects and actor interact.

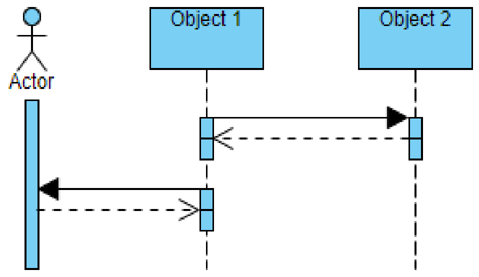


Figure 5. Message Sequence Diagram

4. Case study

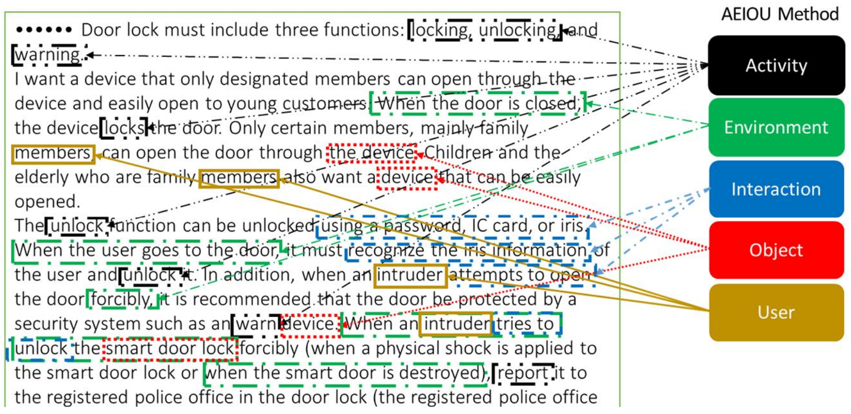
We illustrate the proposed mechanism with the example of a smart door system [11]. We apply the proposed method to this example to show how it actually works with real-time example. Figure 6 illustrates a big chunk of requirements that is written in natural Language

The door lock device that our company wants must be simple to be installed on various types of doors. Door lock must include following functions: lock, unlock, and warn. I want a device that only designated members can open through the device and easily open to young customers. When the door is closed, the device locks the door. Only certain members, mainly family members, can open the door through the device. Children and the elderly who are family members also want a device that can be easily opened. The unlock function can be unlocked using a password, IC card, or iris. When the user goes to the door, it must recognize the iris information of the user and unlock it. In addition, when an intruder attempts to open the door forcibly, it is recommended that the door be protected by a security system such as an warn device. When an intruder attempts to open the door by force, an alarm sound and the security system is activated. It would also be nice if the lock could be easily opened without additional tools. When an intruder tries to unlock the smart door lock forcibly (when a physical shock is applied to the smart door lock or when the smart door is destroyed), report it to the registered police office in the door lock (the registered police office should be nearby the location).

Figure 6. NLR of Smart Door Lock

Step 1. Requirement Analysis by Breaking Requirement Sentence by Sentence

As we proposed above, we divide the NLR into sentence units. And then, Analyzing the requirements of each sentence unit through AEIOU technique to classify the sentence. Figure 7 shows an example that applies to NLR of Smart door lock.



•••••• Door lock must include three functions: locking, unlocking, and warning.
I want a device that only designated members can open through the device and easily open to young customers. When the door is closed, the device locks the door. Only certain members, mainly family members, can open the door through the device. Children and the elderly who are family members also want a device that can be easily opened.
The unlock function can be unlocked using a password, IC card, or iris. When the user goes to the door, it must recognize the iris information of the user and unlock it. In addition, when an intruder attempts to open the door forcibly, it is recommended that the door be protected by a security system such as an warn device. When an intruder tries to unlock the smart door lock forcibly (when a physical shock is applied to the smart door lock or when the smart door is destroyed), report it to the registered police office in the door lock (the registered police office

Figure 7. Classify sentence unit factor by AEIOU method.

We can check Figure 8 for more detail what happens in one sentence unit. First, we took ‘Warn’ as an Activity. It is because warn is the main verb of the sentence. E is situational phase or clause. So, I took ‘when an intruder attempts to break the smart door lock’ as ‘E’ of the sentence. The sentence represents the conditions in which the function of the requirement occurs. We took ‘to break the smart door lock’ as ‘I’ of the sentence. It represents the part where Intruder(User) and Object (The device) interact. ‘O’ is obviously the device. It represents a noun objects except User in the sentence. Object mainly represents the product to be made which is the device in this sentence. Finally, we took ‘Intruder’ as ‘U’ of the sentence. It is a user that tries to break the device which is ‘O’ of the sentence.

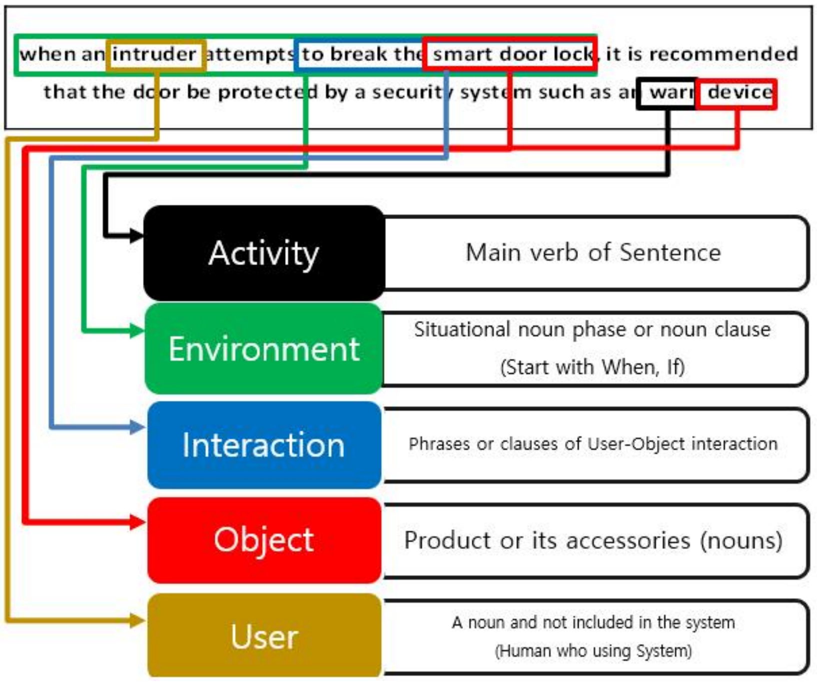

when an intruder attempts to break the smart door lock, it is recommended
that the door be protected by a security system such as a warn device
Activity
Main verb of Sentence
Environment
Situational noun phase or noun clause (Start with When, If)
Interaction
Phrases or clauses of User-Object interaction
Object
Product or its accessories (nouns)
User
A noun and not included in the system (Human who using System)

Figure 8. One sentence analysis by AEIOU method.

Step 2. Redefine requirements with DT technique

Once the requirements are categorized into AEIOUs, they are divided into each element. According to the proposed method above, the requirement sentence is redefined by the requirement redefine rule.

In figure 9, the object, the device, was the subject of the sentence and placed 'Warn' in the main verb position as activity. In order to explain the subject's situation (condition), User placed the device's interaction in the environment prior to the subject. Then, Intruder(User) + Attempt to break(Interaction) + the smart door lock(which is equal to the device) becomes an environment (condition) of the requirement sentence. The form of the redefine rule consists of the main topic after condition of the sentence. When condition tells the environment of the requirement, it specifies certain situation that is non-negotiable for the device.

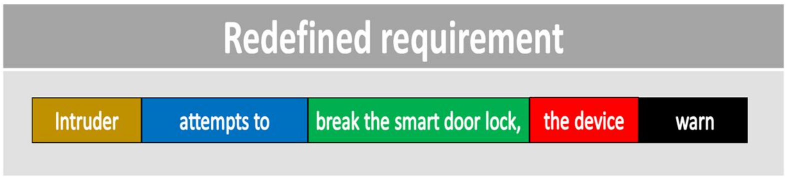


Figure 9. Redefine an example of Smart Door Lock Requirements

Step 3. System design from enhanced requirements specification

As we mentioned above, in step 3, requirements modeling is performed using use case technique. Figure 10 shows use-case diagram for Smart Door Lock System. We put actors from users so that members and intruder become actors of the system. And then we put 'arrows' between actors and use-case scenarios. It represents interaction between activities and users.

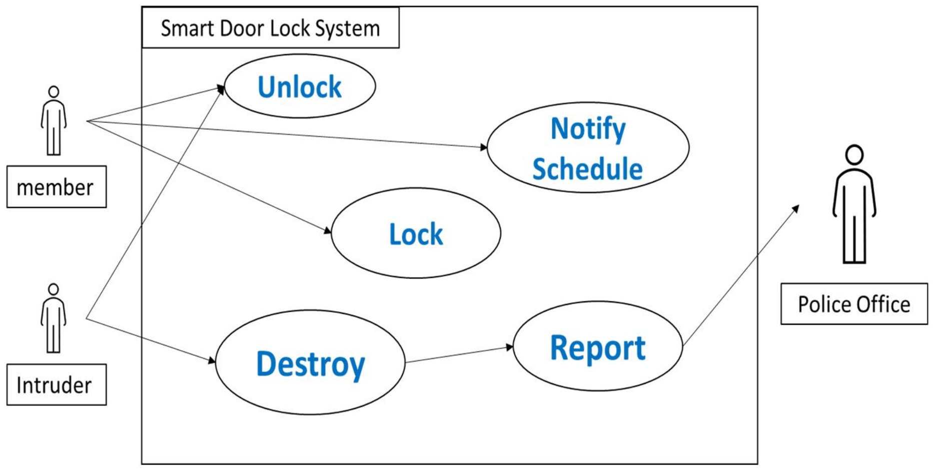


Figure 10. Use-case Diagram for A Smart Door Lock System using

Step 4. Abstract Design with Sequence Diagram

After creating a use case diagram, we learn what functions the system performs. Then we can use sequence diagrams to make abstract designs on how to interact between Actor and Use-case scenarios. When writing a sequence diagram, one pays attention to the movement of information. Figure 11. Show part of the activities of the smart door lock. When the user, intruder, gives an input to the device, the device check the information that user provides for next actions by the confirmation process via database. Because we know clear information who is user or not and what massage brings, we can know how to design the functions of device clearly.

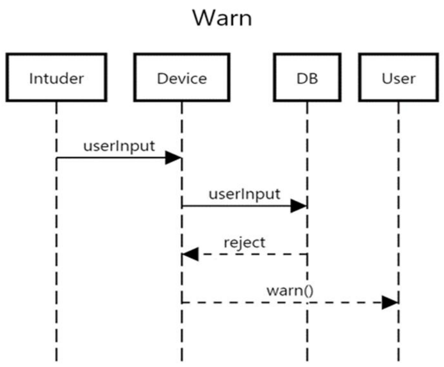


Figure 11. Message Sequence Diagram

5. Conclusion

This paper is a research for redefining ambiguous and unclear requirements by applying design-thinking methodologies to minimize ambiguity in the requirements to improve the quality of the requirements. Also, we expect to minimize the gap between customers, requirement analysts, and developers on understanding of the software product. Furthermore, we expect to improve software quality by semantically clarifying the requirements through the proposed method. In the future, we will conduct research to supplement the limited parts of the proposed methods and automate them as a tool.

Acknowledgement

This work was supported by the National Research Foundation(NRF), Korea, under project BK21 FOUR.

References

[1] Sommerville, Ian (2009). Software Engineering (9th ed.). Addison-Wesley. ISBN 978-0-13-703515-1.

[2] H. J. Kim, D. H. Cho, and T. H. Ahn, "A Study on the Relative Importance of Software Proposal Evaluation Factors Using AHP Technique: Focusing on Comparison between Buyer and Order Holder", *Journal of Information Technology Services*, Vol.16, No. 1, 48-49, 2017.

[3] G. E. Mogyorodi, B. Math., "Requirements-Based Testing - Cause-Effect Graphing", Software Testing Services, 2005.

[4] INTERACTION DESIGN FOUNDATION, *https://www.interaction-design.org/literature/topics/design-thinking*

[5] Creative Thinking and Coding Compilation Committee, Creative Thinking and Coding Engineering, Nosvos, 2018.

[6] C. Seo, "A Study on mapping Software Engineering with Design Thinking mechanism", *2020 Online Spring Conference*, Korea Information Processing Society, p349-351, 2020.

[7] J. Jeong, W. Kim, and R. Kim, "UBAF(User Behavior Analysis Framework) for u-Home Network." *The Journal of The Institute of Internet, Broadcasting and Communication*, Vol. 8, No. 5, pp. 121-127, 2008. ISSN: 1738-4281

[8] B. Davey and C. Cope, "Requirements elicitation–what's missing", *Informing Science and Information Technology*, 5(1), 543–551.

[9] B. Cho, S. Lee. "A Comparative Study of Requirements Analysis Technology using Natural Language Processing and Machine Learning." *Journal of the Korean Society for Computer Information and Information Science*, Vol. 25, No. 7, p27-37. 2020.
DOI: https://doi.org/10.9708/jksci.2020.25.07.027

[10] TSUJI, Kayo. "Implementation of the Writing Activity Focusing on 5W1H Questions: An Approach to Improving Student Writing Performance." *LET Journal of Central Japan*, 28: 1-12, 2017.
DOI: https://doi.org/10.20656/LETCJ.28.0_1

[11] J. Oh, Media Writing, Asian Publishing House, ISBN 978-8-99-400656-7, 2013.

[12] Object Management Group, OMG Unified Modeling Language, *https://www.omg.org/spec/UML/2.5/PDF*, March 2015.

[13] Free UML Diagram Tool. Retrieved, *http://www.uml-diagrams.org/use-case-diagrams.html*