Journal of Theoretical and Applied Information Technology
10th March 2015. Vol.73 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

SOFTWARE QUALITY MEASUREMENT AND IMPROVEMENT USING REFACTORING AND SQUARE METRIC METHODS

[1]FEBY ARTWODINI MUQTADIROH, [2]HANIM MARIA ASTUTI, [3]ARTHA PATRA PRADANA
Department of Information System, Sepuluh Nopember Institute of Technology, Surabaya, Indonesia
E-mail: [1]feby@is.its.ac.id, [2]hanim@its-sby.edu, [3]artha69@gmail.com

ABSTRACT

Software requirement is one of critical factors in a successful software development. Based on some existing researches, a good or poor design of software relies heavily on the quality of software requirements as a definition of software requirements is certainly an early stage in software development. In an IT project, such as a development of School of Social Network (SSN), some problems concerning software quality requirements may occur any time. The changes occurring in the software requirements and the mismatch among the needs, designs, and final result of the project, can lead to poor quality of the software produced. To minimize the problems, it is necessary to measure the quality of software requirements based on SQuaRE Metrics using Refactoring. First, it is to determine the characteristics of software quality requirements. The determination of quality characteristics of SSN is based on an expert experience and eventually sets 4 attributes of software quality requirements that are considered very important, namely: Correctness, Completeness, Consistency and Non-Ambiguity. Second, it is to give a weight on each quality characteristic to obtain the IRQ value. Third, the process of refactoring is conducted to improve the use-case scenarios. And the last step is to re-measure the quality of re-factored software requirement. The early measurements showed that the quality requirements of the SSN reached 39%. Through refactoring process, the improvement of software requirements caused an increase of 6 use cases. And after repair by using refactoring, it increased the quality of the requirements of SSN by 62%. Refactoring is definitely helpful for enhancing the understanding on software requirements without changing the software business process.

Keywords: *Software Requirements, Use Case Scenario, Software Quality, SQuaRE Metrics, Refactoring*

1. INTRODUCTION

In an IT project, an engineering requirement would certainly never miss from software development processes. This is due to the fact that software requirement is a foundation in developing software. Software development process is often referred to as a System Development Life Cycle (SDLC). In SDLC, software requirements will be in a phase of System Analysis/Requirement Definition, pertaining to be the most important phase in a software development [1].

But in reality, the software requirements are often ignored, consequently the quality of software designed is not favorable as expected. Such incidents often occur in a software development process. Ideally, good software is also subject to a good software quality requirement.

To determine software quality requirements, it calls for a quality measurement to be performed. However, a measurement on software quality requirements is often ignored, leading to poor quality of the software requirements and worse output generated anyway.

The main factor affecting the quality of an information system application is identification of requirements covering all aspects of the functionality of an information system that will be designed. The more detailed and comprehensive information in the process of identifying the requirements of the information system, the more capability to cover all functional aspects of an application will be, such as aspects of usability, re-usability, maintainability and other aspects in order to meet the user needs [2].

Researchers found that there is still a developing information system producing applications that are not qualified at all. One of the factors leading to poor quality of applications is the rapidly changing user's needs. But in the paper [3] it was found that

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

there was a less interesting issue that greatly affects the quality of a software development which is in a process of identifying needs (Requirements). A poor quality at the stage of identifying the software requirements will cause a failure to know the aspects of functionality and user's needs. Concrete examples are the identification of requirements on a large scale resulting a lack of clarity on aspects of the functionality in the information system applications that have an impact on lazy requirements, the same activity in a single process (duplicate), and so on. These problems are often encountered in identifying the requirements in the process of information systems development [4].

To minimize the problems above, an improvement of the quality of information systems development is required, especially in the aspect of identifying the requirements of an information system. The methods used in improving the quality of software comprise Refactoring, Requirements Management Plan, and Quality Modelling. However, the most suitable method and focus on the functionality of the system is the Refactoring. The other two methods are focused on all aspects of software developments. Theoretically refactoring according to [5] is a technique to restructure the programming code without changing its functionality. This definition is then adapted with and applied to the stage of requirement identification of an information system development.

In this research, the restructuring of the information on the identification of user's needs adopts refactoring techniques, while the measurement of software requirement complies with the Square metrics standard. This technique will be applied to document of SSN (School Social Network) software requirements as a case study. Accordingly, it is expected that the quality of SSN requirements will increase and be capable of covering all aspects of information system functionality and user's needs that will yield a qualified information system.

2. LITERATURE REVIEW

2.1. Software Evaluation

According to the [6], software quality requirements may be input for a software product quality evaluation. Software evaluation is a package of evaluation technology for measuring software quality characteristics, sub-characteristics or attributes. The package includes evaluation methods and techniques, inputs to be evaluated, data to be measured and collected and supporting procedures and tools. Software product evaluation is a technical operation that consists of producing an assessment of one or more characteristics of a software product according to a specified procedure [6].

For this reason, this study will discuss the evaluation of the software with the aspect of the functionality through the measurement of software requirement quality.

To evaluate and measure the quality of software requirements, this study utilizes SQuaRE matrix which is a standardized evaluation model of ISO / IEC 14598. The more details are described in Section 2.6.

2.2 Software Quality

In a Software Quality according to [2], it is said that qualified software must be in accordance with the requirements specified. In detail [2], it is said that a software quality refers to the suitability of the software that is designed based on requirement specifications.

There are several factors that affect a poor quality of the software. Researchers in [2] classified the causes of the poor quality of the software, including:

1) Faulty definition of requirements

Errors of this type are due to the lack of user understanding for what is actually a necessity. The types of errors that occur are usually as follows:

- Erroneous definition of requirements. This is an error in defining the needs of user;
- Absence of vital requirements. In this sense, the important needs that are not identified in the software;
- Incomplete definition of requirements. This relates to the identification of the needs on software that does not include the functionality aspects in the user's needs;
- Inclusion of unnecessary requirements, which means the needs that are not needed;

2) Client-developer communication failures

Failure to identify the needs of the software is usually caused by poor communication between the client and the developer. One example of poor communication between the developer and the client is less response of a developer to the changing needs of software that is delivered by the

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

client, which results in the quality of the software that will be developed later.

3) Deliberate deviations from software requirements

In some cases, developers usually do not follow the terms agreed before that cause errors in software development. Developers sometimes use a module in the previous project to be applied to software development projects being executed. Without any further communication, this will affect the quality of the software to be developed.

4) Logical design errors

Errors in the software could occur during design phase. The design is prepared by those who are lack of competence in a software designing. An error usually occurs in the process of defining the workflow of system to be created.

5) Coding errors

These errors occur in writing line of code when a programmer does not understand the design of the software. At last, there is an incompatibility between the code and the design that will affect the quality of software.

6) Non-compliance with documentation and coding instructions

A software developer generally has his own standards and procedures in developing software. But in a software development project, this will only be a bad thing for the long term at the time of repairing due to a mismatch between the documentation and coding between the developers and the project team.

7) Documentation error

Errors in the software documentation will make the development process more difficult, including software maintenance process conducted by a team of software maintenance. Less integration between software requirements document and design document will influence in the next software development.

Another impact caused by the fault software documentation (user guide documentation) is to confuse the end user to operate the software. The general errors are:

- Some mistakes in describing the instructions to use the software;
- Provision of a guidance that is not on the application.

In this study, researchers focused on the "Faulty definition of requirements" problems, where the identification of software requirements is the main and very important aspect to determine the quality and the failure of a software.

2.3 Requirement Analysis

The main purpose of software requirements analysis is to obtain and to identify the software needs and the conditions to be satisfied during the software development. According to [7], software requirements analysis is one of the factors determining the success of software development projects.

Conceptually, there are three main activities in software requirement analysis including:

1) Eliciting Requirements

Identify the software needs taken from several sources, including project documentation (Project Charter), business process documentation, and stakeholder interviews. These activities are commonly referred as Requirement Gathering.

2) Analysing Requirements

Determine whether the requirements have been already clear, complete, consistent, and unambiguous.

3) Recording Requirements

Document the requirements that have been obtained in the previous analysis activities. The document contains use case, user stories, and specification process.

2.4 Use Cases

According to [7], the use of case describes the interaction between the actors and the system is interconnected. The use case is a part of use case diagram. Use case diagrams are very important in explaining what is done by a system then specifying the work through Flow of Events. The flow of Events describes a use case in a clear definition and describes how the use case starts and ends. The flow of events is commonly known as the use-case scenarios. According to [8], it is said that the scenario is one way of representing the requirements. As the scenario describes a system from a user perspective, it focuses on user-system interaction.

Journal of Theoretical and Applied Information Technology
10th March 2015. Vol.73 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

2.5 Refactoring

According to [5], refactoring is a technique for restructuring the code without changing the functionality of the programming. The aim of refactoring is to improve understanding of programming code and reduce the complexity that impacts on improvement to the maintainability of the programming. Developers usually called the term of "code smell" or "bad smell" during perform the refactoring. An example is a programming method which quite a lot duplicated with another method.

There are two benefits derived from the refactoring methods, namely:

1) Maintainability. Refactoring would make the code easier to understand especially when make improvements to the bugs found in the software.
2) Extensibility. Refactoring will make easier to expand on the capabilities of the software.

Yet, refactoring method is not only used to improve the program code, but also can be used to improve the definition of software requirements in the early stages of analysis, which is the scenario. The scenarios, according to [8], describe sequential interaction to define the specification of requirements that are part of a series of software requirement quality measurement. In the paper [3], it elaborated what should be considered before determining the use case scenarios that can be repaired using refactoring. These things are as follows:

1) **Large Requirements.** This is a condition in which a use case to try to accommodate multiple functions or objectives or a use case to have excessive alternative flows;
2) **Complex Conditional Structure.** This is a condition in which a use case to have complex structure or the software needs requires some other software needs to be a unity of good software needs. Other arising conditions are when there is nested conditionals;
3) **Lazy Requirement.** This is a condition in which the function or a role of a software requirement to have vague impact on the system. In addition to that, this state also indicates a condition in which a software requirement does not accommodate all of the activity intended or incomplete requirements;
4) **Naming Problems.** This is a condition in which a naming the software requirement does not refer to a concept that has been determined, or a condition where the same name is the used for different concepts (Ambiguous);
5) **Duplicate Activities.** This is a condition in which the same software needs to have duplicates in different places in the software requirements document. An example is where a main flow or alternative flow is repeated on a software requirement.

2.6 SQuaRE Matrix [9]

SQuaRE or Software Product Quality Requirement and Evaluation is a strategy that includes criteria for the specification of quality requirement and the evaluation to measure the quality of software requirement.

SQuaRE Matrix is a generic model of an evaluation process, supported by the quality measurements from ISO/IEC 14598. The model is presented in Figure 1 that specifies four major sets of activities for an evaluation:

1) Establishment Evaluation Requirements
2) Specification of Evaluation
3) Design of Evaluation
4) Execution of Evaluation.

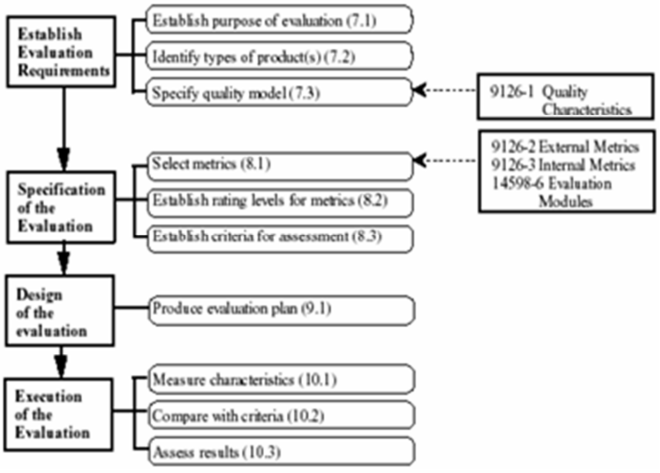

Figure 1. Evaluation Process of ISO/IEC 14598

Journal of Theoretical and Applied Information Technology
10th March 2015. Vol.73 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

3. RESEARCH METHOD

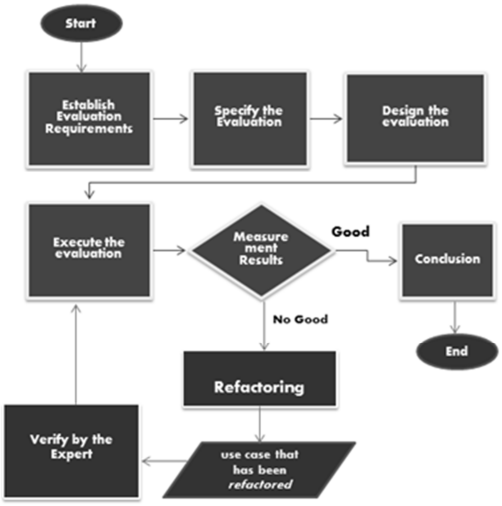

Figure 2. Research Methodology

Methods in research is needed to guide the completion of the research process in order to run as directed and systematic. Here is an overview of research methods used refer to the SQuaRE Matrix [9]:

3.1 Establishment Evaluation Requirements

This stage is the initial stage of measuring the quality of software requirements. The data have been obtained from a meeting with the SSN application developers then will be analyzed by the experts. Analysis by expert conducted to determine the quality of the use case scenarios of SSN application based on 4 criteria of quality (completeness, correctness, consistency, non-ambiguity) are described in the paper [8] and whether those use case scenarios to have refactoring oppurtunites. To determine the quality of the use case scenario, some verification questions need to be prepared for the expert which refers to the four criteria of the software requirements quality.

3.2 Specification of Evaluation

After obtaining the use case scenario even if contained any less qualified, then the next step is determining the weight of each criterion and determine the rating level of each use case scenario. The results of this process is assigned that the value of 1 means the use case scenario is qualified, while the value of 0 means the use case scenarios do not satisfy the quality. Furthermore, the weight of each quality criterion given by expert is based on the urgency is as follows:

- Correctness must achieve 90%
- Completeness must achieve 75%
- Consistency must achieve 75%
- Non-Ambiguity must achieve 70%

The result of weighting and rating level is going to be used for the measurement of the quality of use case scenario in SSN application.

3.3 Design of evaluation

The aim of this stage is to create an evaluation plan document that contains a process of the quality measurement for use case scenario in SSN application.

3.4 Execution of evaluation

This stage is the last stage of quality measurement process of the use case scenario in SSN application. Measurements were created using weights and rating level put into the formula for calculating the quality. The results of these calculations will later determine whether a use case scenario qualified or not.

3.5 Refactoring

The result of quality measurement in use case scenario will be the input for this stage. The results are processed using refactoring. There are 5 ways to make improvements using refactoring, include:

- Extract Requirements

  This process is conducted when there is information about large-scale software requirements which can be divided into 2 or more new software requirements (in the same context). According to the paper [3] The need of software contains a lot of important information and difficult to understand, so then it will be not easy to find the required information quickly.

- Rename Requirements

  Giving the name of the software requirements adjusted to the context of the needs of the software. Giving a good name will facilitate communication and understanding of the system and the use of abstraction in general vocabulary that will facilitate the development team [3].

- Move Activity

  This process focuses on improving the modularity and balancing the activity in defining software

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

requirements. This process could be happened. This process can also occur during the requirements extraction process (Extract Requirement) by moving activities to the desirable needs. [3] said that the increased modularity in software requirements will lead to a better understanding of the system in the long term.

- Inline Requirement

  The goal of this process is to reduce the complexity of software requirements by combining several existing requirements. If a software requirement is not important enough to be used, then the developer can perform inline - merging to the other software requirements. Each software artefact requires time and resources to understand and maintain the software [3].

- Extract Alternative Flows

  This process is performed when the information flow on the software requirements do some scenarios at the same time resulting in the accumulation of information that affects the understanding of requirement responsibilities and information flow are minimal and difficult. Alternative scenario is the right way to manage the complex information flow to the certain structured conditions [3].

3.6 Verification of Use Case Scenario to the SSN Application

After repairing the use case scenarios using refactoring, the next step is to verify to the expert. A verification process is similar to the process at the beginning before refactoring. The aim of the verification process is to determine how the quality of the use case scenario of SSN application when assessed from four quality criteria, namely: completeness, correctness, consistency, and non-ambiguity.

4. RESULT AND DISCUSSION

4.1 Use Case Scenario Selection based on Literature Review

The selection of a use case scenario is focused on the use case that has the opportunity to be re-factored. Based on observations and analysis fitting to the description in paper [3], the use case scenarios are selected to be repaired by refactoring as presented in Table 1:

Table 1: Comparison of Accuracy, coverage, and covacc between CBS and modified CBS algorithm with FEAT and FSGP algorithm

Literature Review	Use Case are Problematic
Improving the Quality of Requirements with Refactoring [3], [8]	UC-05 View Profile UC-06 View feeds of Profile UC-12 Friendship Confirmation UC-26 Attendance Confirmation UC-35 View the Study Report

4.2 Use Case Selection based on Expert Judgment

Furthermore, to better ensure a use case scenario which should be improved, the next step is to use an expert judgment method. This method is conducted by contacting several experts who have experience to analyze the use case scenario of SSN application. Selected 3 experienced expert to analyze use case scenarios to do refactoring opportunities.

Table 2: Use Case Selection Using Expert Judgment

Expert 1	Expert 2	Expert 3
UC-04 Send Comment	UC-04 Send Comment	UC-04 Send Comment
UC-05 View Profile		
UC-06 View feeds of profile	UC-06 View feeds of profile	UC-06 View feeds of profile
UC-12 Friendship Confirmation	UC-12 Friendship Confirmation	UC-12 Friendship Confirmation
UC-26 Attendance Confirmation	UC-26 Attendance Confirmation	UC-26 Attendance Confirmation
UC-27 Comment to the Event	UC-27 Comment to the Event	UC-27 Comment to the Event
UC-35 View the Study Report	UC-35 View the Study Report	UC-35 View the Study Report
UC-36 Send a Liaison Book	UC-36 Send a Liaison Book	UC-36 Send a Liaison Book

Journal of Theoretical and Applied Information Technology
10th March 2015. Vol.73 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4.3 Quality Measurement of Use Case Scenario

Measuring the quality of use case scenario will generate 2 conditions; the quality prior to refactoring and the quality after refactoring. The measurement processes consist of interviews, calculations and inferences. Table 3 represents the results of the interview processes as follows:

Table 3: Recap of the value of the Software Requirement Quality to the Use Case Scenario for each Quality Characteristic

Use Case Scenario	Quality Characteristic	Value
UC-04 Send Comment	Correctness	1
	Unambiguity	1
	Completeness	1
	Consistency	0
UC-06 View feeds of profile	Correctness	0
	Unambiguity	0
	Completeness	0
	Consistency	1
UC-12 Friendship Confirmation	Correctness	0
	Unambiguity	0
	Completeness	1
	Consistency	0
UC-26 Attendance Confirmation	Correctness	0
	Unambiguity	0
	Completeness	1
	Consistency	1
UC-27 Comment to the Event	Correctness	1
	Unambiguity	1
	Completeness	1
	Consistency	0
UC-35 View the Study Report	Correctness	0
	Unambiguity	0
	Completeness	0
	Consistency	1
UC-36 Send a Liaison Book	Correctness	1
	Unambiguity	0
	Completeness	1
	Consistency	1

The value [10] generated by the expert is based on questionnaire. An expert judged 7 Use Cases in accordance with four quality criteria. Each quality criterion has a value of 1 and 0. The value of 1 means the use case scenario is appropriate with the quality criteria, while the value of 0 means the use case scenarios do not meet the quality criteria.

Measurement of the 4 characteristic qualities will produce Individual Requirements Quality (IRQ) [10]. The IRQ presented the quality of each characteristic as the average of the overall value of the quality characteristic that has been obtained from the questionnaire.

1) IRQ to measure the Correctness:
$$\frac{\sum_{i=1}^{n} Correctness}{n}$$

2) IRQ to measure the Completeness:
$$\frac{\sum_{i=1}^{n} Completeness}{n}$$

3) IRQ to measure the Consistency:
$$\frac{\sum_{i=1}^{n} Consistency}{n}$$

4) IRQ to measure the Unambiguity:
$$\frac{\sum_{i=1}^{n} Unambiguity}{n}$$

Table 4: The Result of the Software Requirement Quality

Quality Characteristic	Total of Value 1	Total of Value 0	IRQ
Correctness	3	4	0,43
Completeness	5	2	0,71
Consistency	4	3	0,57
Unambiguity	2	5	0,29
		Total	**2,00**

Table 4 represents the results of IRQ. The next calculation is continued by multiplying each value of the quality characteristic with the weight given by the expert as shown in Table 5.

Table 5: The Result of Individual Requirement Quality Equipped with Weights

No	*Quality Characteristic*	IRQ	Weight	IRQ * Weight
1.	**Correctness**	0,43	0,95	0,4085
2.	**Completeness**	0,71	0,75	0,5325
3.	**Consistency**	0.57	0,75	0,4275
4.	**Unambiguity**	0,29	0,70	0,203
			Total	**1,5715**

Journal of Theoretical and Applied Information Technology
10th March 2015. Vol.73 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Figure 3 shows the comparison between the perfect amount of quality characteristics with measurement results:

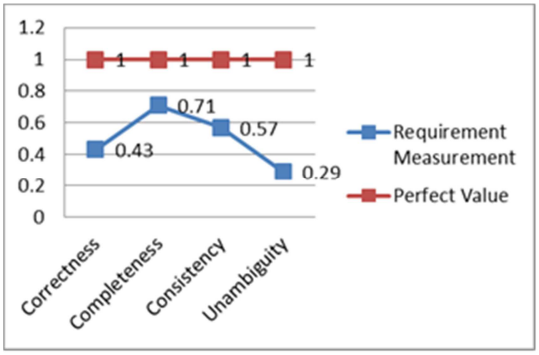

Figure 3. Graph Representation of SSN Requirement Measurement

After getting the value of the result of multiplying with the weight on each quality characteristics, the next step is identifying the average of each quality characteristic. In this way obtained the results of Requirement Quality as follows:

$$\textbf{\textit{Requirement Quality}} = \frac{\sum IRQ * Weight}{n} = \frac{1{,}5715}{4}$$

$$\textbf{\textit{Requirement Quality}} = \mathbf{0.392875}$$

The result of Quality Requirements is 0.392875, which means that the quality of the SSN application requirements only reached 39%.

4.4 Refactoring Process

Refactoring process is based on the analysis and measurement of the use case scenario on the SSN application. The following is an example of the results of the analysis of the use case scenario on the SSN application that is problematic then adjusted for corrective action according to refactoring method. Table 6 presents the use case of Send Comment to be refactored.

Table 6: Refactoring the Use case "Send Comment"

UC-04 Send Comment	
Problems	The use of the word customer in in a use case narative inconsistent
Opportunities	Naming Problem
Refactoring	Rename Requirement
Solution	Change the word of customer according to the similarity in the other use case scenarios
Motivation	Paper [3] explains that the use of a good name would make a better understanding and an easier way communication for the development team.
Mechanism	1) Select the requirement to be corrected 2) Change a part of the requirements that need to be changed 3) Customize the content of the requirements that have been changed
Result	Changing the word of customer becomes user according to the existing content on the UC-04 Send comments

Another example is also presented in Table 7 which presents the use case of View feeds of profile to be refactored.

Table 7: Refactoring the Use case "View feeds of profile"

UC-06 View feeds of profile	
Problems	This use case should not be used as the main use case. It is rather more appropriate as an alternative flow of a main use case
Opportunities	Lazy Requirement
Refactoring	Inline Requirement
Solution	Move the requirement description of UC-06 into a use case scenario of UC-05 View Profile
Motivation	Paper [3] explains if a use case existence is not suitable to be used as the main use case, then developers may combine (merge) the use case to another use case.
Mechanism	1) Copy all activities, including the prerequisite condition described in the existence use case to the use case to be merged 2) Update all content or other use cases affected by this process 3) Remove the information referring to the use case scenario that is to be merged
Result	The activity of UC-06 is becoming the alternative flow to the UC-05 View Profile. On the profile page, user able to see the activities that have been done by him/herself or friends such as the activity of seeing status, seeing wall posts, etc.

Table 8 contains a summary of the problems in the use case scenarios of SSN and the description of the refactoring process:

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 8: Problems of SSN Requirement and the Result After Refactoring

Use Case Identification Before Refactored	Problems	Use Case Identification After Refactored
UC-04 Send Comment	Naming problem to the actor	UC-04 Send Comment
UC-06 View feeds of profile	Included in the use case alternative	UC-05 View Profile
UC-12 Friendship Confirmation	Use case is too general and should be subdivided into several use cases	UC-13 Accept Friendship Confirmation UC-14 Reject Friendship Confirmation
UC-26 Attendance Confirmation	Use case is too general and should be subdivided into several use cases	UC-27 Presence Confirmation UC-28 Absence Confirmation
UC-27 Comment to the Event	Naming problem to the actor	UC-30 Comment to the Event
UC-35 View the Study Report	Use case is too general and should be subdivided into several use cases	UC-36 View the Study Report UC-37 View the grades of kindergarten UC-38 View the Report of Personal Development UC-39 View the grades of Cambridge UC-40 View the grades of Subjects
UC-36 Send a Liaison Book	Naming problem to the actor	UC-41 Send a Liaison Book

Here are 6 new use case after refactoring process:

- UC-14: Reject Friendship Confirmation
- UC-29 Absence Confirmation
- UC-37 View the Grades of Kindergarten
- UC-38 View the Report of Personal Development
- UC-39 View the Grades of Cambridge
- UC-40 View the Grades of Subjects

4.5 Quality Re-Measurement of Use Case Scenario

The next step is to re-measure the quality of the use case scenarios that have been refactored based on four criteria of quality of Correctness, Completeness, Consistency and Non-ambiguity. The measurement is conducted through questionnaires to experts. The experts judged 7 Use Case based on 4 quality criteria. Each quality criterion has a value of 1 and 0. The value of 1 means the use case scenario is appropriate with the quality criteria, while the value of 0 means the use case scenarios do not meet the quality criteria.

Table 9 represents the results of the expert questionnaire on the quality characteristics of the use case after being refactored.

Table 9: The Result of the Software Requirement Quality After Refactoring

Quality Characteristic	Total of Value 1	Total of Value 0	IRQ
Correctness	6	1	0,86
Completeness	7	0	1
Consistency	5	2	0,71
Unambiguity	4	3	0,57
		Total	**3,14**

In the process of measuring the quality characteristics obtained the Individual Requirements Quality (IRQ). IRQ presented the quality of each characteristics by calculating the average of the overall value of the quality characteristic has been obtained from the questionnaire.

Having obtained the results of IRQ, the next calculation is continued by multiplying each value of the quality characteristic with the weight given by the expert as shown in Table 10:

Journal of Theoretical and Applied Information Technology
10th March 2015. Vol.73 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 10: The Result of Individual Requirement Quality Equipped with Weights After Refactoring

No.	*Quality Characteristic*	IRQ	Weight	IRQ * Weight
1.	Correctness	0,86	0,95	0,817
2.	Completeness	1	0,75	0,75
3.	Consistency	0,71	0,75	0,5325
4.	Unambiguity	0,57	0,70	0,399
			Jumlah	2.4985

After getting the value of the result of multiplying with the weight on each quality characteristics, the next step is looking for the average of each quality characteristic. In this way obtained the results of Requirement Quality as follows:

$$\textbf{\textit{Requirement Quality}} = \frac{\sum IRQ * Weight}{n}$$

$$= \frac{2.4985}{4} = 0.624625$$

Based on the results of the software requirements assessment in the use case scenario of SSN application that has been corrected using refactoring, the result of the Quality Requirements measurement showed an increase up to 62%.

Figure 4 shows the comparison between the software requirement quality of the use case scenarios before repaired with the software requirement quality of the use case scenario after refactoring.

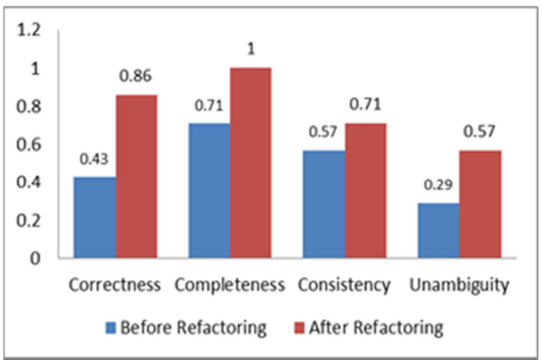

Figure 4. Graph Representation of SSN Requirement Mesurement Before and After Refactoring

5. CONCLUSION

5.1 Conclusion

The conclusion that is derived from this study is as follows:

1. Based on the measurement of the initial conditions of use case scenarios of SSN application it generates a quality of software requirement by 39%, showing that the quality of the SSN application before repaired is still low.
2. Based on the refinement using refactoring, the quality of the use case scenario of SSN application increases up to 62%.
3. By using refactoring, the use case scenario of SSN application grows to six use cases: Reject Friendship Confirmation, Absence Confirmation, View the grades of kindergarten, View the Report of Personal Development, View the grades of Cambridge, View the grades of Subjects.

5.2 Suggestion

The suggestions expected to be developed in the future are:

1. Refactoring process is completed when a programming has been tested using a test case to ensure that the programming code has been re-factored does not change the behavior of the overall system.
2. Since the refactoring is based on the document that cannot be tested (test case document), further research is needed to ensure that the refactoring will not change the behavior of the overall system considering the goal of refactoring is to facilitate a use case to be well understood.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

REFERENCES

[1] I. Sommerville, *Software Engineering*, 9th edition. Boston, Massachusetts: Addison-Wesley, 2010, pp. 27–74.

[2] D. Gallin, *Software Quality Assurance from Theory to Implementation.* Edinburgh Gate : Pearson Education, 2004.

[3] J. Ramos, Ricardo; Piveta, Eduardo K; Castro, Jaelson; Moreira, Ana; Guerreiro, Pedro; Pimenta, Marcelo S; Price, R. Tom; Araujo, "Improving the Quality of Requirements with Refactoring," in Simposio Brasileiro de Qualidade de Software, 2009.

[4] D. Firesmith, "Common Requirements Problems, Their Negative Consequences, and the Industry Best Practices to Help Solve Them," *J. OBJECT Technol*., Vol 6, No. 1, pp. 17–33, 2007.

[5] J. Kerievsky, *Refactoring to Patterns*. Addison-Wesley Signature, 2004.

[6] ISO/IEC and I. S. 25030, "Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Quality requirements," BS ISO/IEC 25030:2007. 2007.

[7] I. Jacobson, "Use Cases and Aspects - Working Seamlessly Together," *J. OBJECT Technol.* Vol. 2, No. 4, pp. 7–28, 2003.

[8] M. Glinz, "Improving the Quality of Requirements with Scenarios," in *Proceedings of the Second World Congress for Software Quality* (2WCSQ), 2000, pp. 55–60.

[9] A. Suryn, Witold; Abran, "ISO/IEC SQuaRE. The second generation of standards for software product quality," in IASTED, 2003, pp. 1–11.

[10] A. M. Essado, Marcelo; Ambrosio, "A Requirement Evaluation Metric Applied on the ITASAT-1: A Small Technological Satellite," 2012.