Requirements Defects Techniques in Requirements Analysis: A Review

M. Kamalrudin[1], L. L. Ow[2] and S. Sidek[2]
[1]Innovative Software System and Service Group, Universiti Teknikal Malaysia Melaka, Hang Tuah Jaya, 76100 Durian Tunggal, Melaka, Malaysia
.[2]Universiti Teknikal Malaysia Melaka, Hang Tuah Jaya, 76100 Durian Tunggal, Melaka, Malaysia.
massila@utem.edu.my

***Abstract*— Defects existing in the systems due to poorly identified requirements defects are viewed as the major factors leading to system failure, especially if the requirements defects are not identified or addressed until at the later stage of software development life cycle. In response to this, several attempts have been made to identify defects during the requirements analysis process. This paper presents a review of the various techniques to handle requirements defects in the requirements analysis activity. These techniques are categorised into four categories, namely reading, inspection, analysis and automated tool. It was found that these techniques have different focus and lack of emphasis on the needs of the industry. This study provides the basis for future research aiming at developing an approach to automate the process of requirements defects handling.**

***Index Terms*—Requirements Analysis; Requirements Defects; Requirements Defects Techniques.**

I. INTRODUCTION

Requirements engineering, conducted at the earliest stage in Software Development Life Cycle (SDLC) is considered as the most important stage. Requirements are pervasive to affect the continuous activity throughout the process of requirements engineering and even SDLC [1]. It comprises of four different activities, namely requirements elicitation, requirements analysis and validation, requirements documentation, and requirements management. Requirements elicitation is the first stage of requirements engineering which aims to elicit requirements through communication with the stakeholders. The second stage is the requirements analysis and validation that aims to detect and resolve requirements conflict raised between the stakeholders. The third activity in requirements engineering is requirements documentation, which focuses on defining what and how the system should be built. At this stage, all the agreed functional and non-functional requirements are documented. The last activity is the requirements management, whereby the requirements are coordinated, scheduled and documented. Throughout the process, the requirements are traceable and manageable in order to manage any changes made on the requirements [1].

Requirements engineering process is a human-based activity as it relies much on human decision-making. Hence, mistakes are common occurrences in the requirements engineering process. In this case, it is vital to ensure high quality of requirements in order to avoid or reduce the likelihood of propagating the defects to the subsequent phases of software development life cycle. Fixing defects at the early stage of requirements engineering process is easier, less expensive and less effort for rework [2]. For these reasons, we thought it is most appropriate to begin requirements defect detection in the early stage of SDLC. This paper reports a comparative study on the existing techniques or approaches to handle requirements defects at the requirement analysis stage. It is found that most of the defects checking approaches start at the stage of requirements analysis in the requirements engineering. We would like to review the current requirements defects techniques in the requirements analysis phase.

The remainder of this paper is structured as follows: Section II elaborates on the definition of requirements defects and Section III portrays the literature review. Section IV presents findings of the comparative analysis and lastly Section V concludes our finding.

II. REQUIREMENT DETECT

Defects are considered, by IEEE Standard Classification of Software Anomalies, as an imperfection or deficiency in specified software whereby the requirements and specifications from the client are not met [3]. Defects are normally needed to be either enhanced, repaired or rework before or after software release. Defects are usually assumed to be appeared in the line of codes. Besides, defects arise from inconsistencies or contradictions within or between the requirements, according to Blackburn et al. [4]. In our context of requirement defects, requirements defects are found in the requirements specification instead of the line of codes. Any imperfection, undesired or vague requirements statements is defined as requirements defects. For example, unstated requirements from the client, tacit requirements between developers and client, misunderstood requirements and etc. are the typical examples of reported requirements defects [5].

According to the study conducted by author Chen and Huang [6], requirements issues are included in the top 10 higher severity problems in software development, as shown in Table 1 below. Besides, study also showed that the effect of requirements defects is more severe and labour intensive due to the propagation since the early stage of software development life cycle and affect the sequential stage. Hence, software development life cycle is developed based on the wrong foundation or interpretation of requirements [7].

Table 1
Top 10 Severity Problems in Software Development [6][8]

#	Software Development Factors	Problem Dimension
1	Inadequate of source code comments	Programming Quality
2	Documentation obscure/untrustworthy	Documentation Quality
3	Changes not adequately documented	Documentation Quality
4	Lack of traceability	Documentation Quality
5	Lack of adherence to standards	Programming Quality
6	Lack of integrity/consistency	Documentation Quality
7	Continually changing requirements	Systen Requirements
8	Frequent turnover within the project team	Personnel Resources
9	Improper usage of techniques	Programming Quality
10	Lack of consideration for software quality requirements	Systen Requirements

III. Current Techniques Of Handling Requirements Defects

According to Brykczynski [9], defects found in the testing phase are usually traceable to requirements and design flaws which can be detected earlier. Defects in the requirements and design stage are relatively inexpensive and easy to rectify. Requirements defects are the most expensive to repair if they are found in the later stage of software development life cycle as they are compounded by having to undo the work done which based on the wrong interpretation of the false foundation. The cost of fixing the requirements defects in the early stage of software development life cycle is usually negligible.

There are reviewers who identify potential requirements defects in the requirements documents by adopting a particular reading technique. There are a few techniques in supporting this activity that has been proven effective. Researchers have agreed that the choice of reading techniques has directly impact upon the inspection [10]. Reading techniques are generally classified into two different categories which refer to systematic reading technique and non-systematic reading technique, as shown in Figure 1 below.

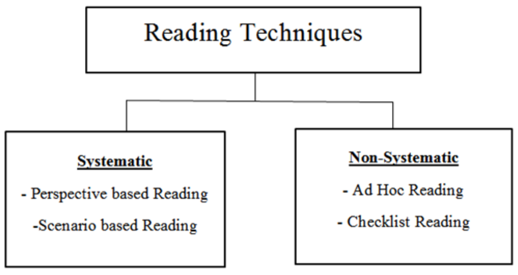

Figure 1: Different Types of Reading Techniques

As shown in Figure 1, the systematic reading technique is a highly explicit structural approach, while the non-systematic reading technique is an intuitive approach. Systematic reading technique, including the perspective based reading and the scenario based reading, provides a set of structural instructions for the reviewers and explains how to find requirements defects by adopting perspective based reading. On the other hand, the non-systematic reading technique, including ad hoc reading and checklist reading, do not have any specific framework for requirements defects detection. As such, it does not offer help in finding defects or support to the reviewers.

The scenario based Reading (SBR) was developed to identify defects in requirements documents. In SBR technique, requirements defects are classified and a set of questions is developed for each of the defect classes. In this case, scenario is referred to a collection of procedures for detecting particular types of requirements defects. The scenario is also developed and focused on specified viewpoints. The reviewers are required to answer the questions based on a specific scenario. Researchers compared the SBR to ad hoc reading and checklist reading techniques by conducting a few experiments in order to confirm the findings[11][12]. The conducted experiment shows that Scenario based Reading had a higher detection rate in comparison to ad hoc and checklist approaches. In addition, they also found that the checklist reading method is no any better than ad hoc reading method.

On the other hand, the perspective based Reading (PBR) is an enhanced version of scenario-based reading, which focuses solely on the point of view or the needs from the stakeholders. Each scenario consists of a set of questions and is developed based on the viewpoint of the stakeholders. The reviewers read the requirements documents from a particular viewpoint with a physical model for analysis in order to answer the questions based on the viewpoint. PBR is expected to reduce any existing gaps or any overlapping between the reviewers during the inspection process. It is claimed to be more effective in finding defects in requirements documents than other less structured reading inspection methods since it is considered as systematic, goal oriented, customable, focused and transferable in training session[13].

Ad hoc reading technique, which is classified as the non-systematic reading technique, is totally different from the rest of the three reading techniques. It collects merely very general viewpoints from the reviewers in identifying requirements defects. There is neither procedure to adopt nor training required for ad hoc reading technique. In this case, reviewers are required to use their own knowledge to identify requirements defects in requirements documents. Also, there is no support given to the reviewers in ad hoc reading technique.

Checklist reading is a more systematic technique in comparison to ad hoc reading technique. It provides a list of questions or predefined issues that needs to be checked during inspection process. Checklist reading techniques aims to define the responsibilities of the reviewer and guide them in identifying defects. However, checklist reading technique usually focuses on questions that aid the reviewer to identify the major defects.

Despite the reading techniques stated above, the most effective process for detecting the defects across all stages in software development life cycle is inspection [9] as manual approach. Inspection technique was developed by Fagan in IBM [14], in 1972, aiming to identify defects. By identifying defects, the process of inspection targeted at reducing the costs and improving the quality. The inspection technique was initially applied to hardware logic then to software logic design and code. It was then been applied to the rest of the process in software development life cycle including requirements phase.

Inspection is easy to implement and is claimed to be a highly effective method for requirements defects detection [2]. Inspection is a manual review of requirements adopting formal review procedures in a group setting. In addition, inspection is known as structured walkthroughs. There are four crucial aspects to be considered in the method of requirements formal inspection namely a well-defined group having an assigned role for each of the requirements inspectors, a checklist for requirements inspection process, an agenda regarding how the requirements inspection will be carried out and lastly a procedure for reviewing the conclusion drew by the inspection team.

Figure 2 below shows the dramatic reduction of the number of defects found throughout the different stages when inspections were deployed. The number shown in the parentheses are the number of defects without inspections. For example, the defects found in the requirements stage were initially 20, but it was reduced to 5 defects after applying the inspections per KLOC, thousands (Kilos) of Lines of Codes. In addition, the amount of rework to correct requirements defects is significantly reduced and productivity is increased simultaneously. Overall, the methods of inspections reduce the cost of software development, increase the quality of software and improve the productivity and the quality of the decision making from the management [10].

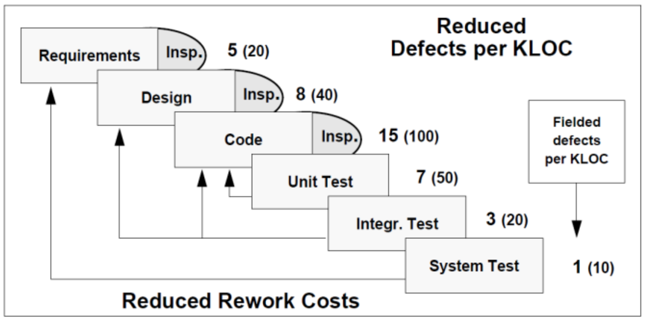

Figure 2: Profile of Defects with Inspections [9]

Despite the advantages of adopting inspections on identifying requirements defects, there are some reasons why inspections are not widely adopted. For example, inspections method required upfront cost for building inspections infrastructure. Some organizations would rather adopt informal review process on identifying requirements defects in order to save cost.

Another experiment carried out by using N-fold inspection method for fault detection in user requirements documents instead of formal inspection method [2]. N-fold inspection method adopts N number of independent efficient teams in identifying defects that might not be found by just a single team. This approaches expected to find different types of defects by different teams of inspection. In their experiment, nine independent teams were asked to locate as many as requirements defects in the given user requirements document by applying N-fold inspection method. The result gained from their experiment was favorable to the initial results reported by Martin and Tsai [15]. In fact, the N-fold inspection method originated from Martin and Tsai was applied on requirements analysis of mission critical system. The N-fold requirements inspection method produces a significant improvement in locating and correcting requirements defects in user requirements document in comparison to the common traditional requirements inspection method. However, requirements faults still found to be occurred in the later stage of development since the basic requirements inspection techniques do not identify or detect all the defects presented [16]. This fault slippage motivates researchers to investigate improvement on the fault detection process. Other researchers had been investigating on various supporting mechanisms as added quality improvement process

Researchers [17] presented a study on Fault Tree Analysis (FTA) to identify potential hazards, fault and failures from a hardware system. FTA is found to be effective in mitigating the risks of potential faults and failures of the system. FTA has been extent for the application in the field of software called Software Fault Tree Analysis (SFTA) due to the effectiveness in mitigating risk. SFTA used to model the intrusion is a backward search. Software Fault Tree Analysis includes the information in a tree diagram of events and logic gates leading to possible hazards. SFTA is recommended to be applied to the requirements and design stage of software development. The main objective of applying SFTA in requirements is to identify weaknesses that lied in requirements specification. Weak requirements are modified or additional requirements to be included in order to eliminate the weak requirements. The other objective of applying SFTA is to identify the requirement that has a direct effect on the system safety issue. The requirements will be able to trace via requirement traceability matrix throughout software development life cycle once the requirements with safety considerations are identified. However, the study focused only on software design, in particular OODs using UML, instead of requirements stage. Their study is still immature and at the very basic stage of investigating application of SFTA on the design stage. They adopted simple design for illustration of SFTA concept and there is no significant result on the usefulness of SFTA on OOD so far.

In the same context of SFTA, researches [18] adopted SFTA in analyzing the intrusion domain to identify and verify the requirements for Intrusion Detection System (IDS). There have been no separate requirements specifications created in their IDS prototypes. The intrusion fault trees are interpreted as specifications of the events combinations that must be detected instead of requirements specifications. SFTA model of intrusion describes requirements indirectly for the design of IDS and assists in verification process. A SFTA related aspect development is considered tedious, detailed work and required expert analysis which is hard for automation. It has to be paired with machine learning approaches in order to automate the development of SFTA.

In the year of 2001, Blackburn et al. [4] described a model based verification approach for locating and correcting requirements defects in the early stage of the development process. Their approach refers to Test Automation Framework (TAF) which integrates available model development and test generation tools in order to support defect prevention and test automation. They believed that automate test generation eliminates the common manual and error prone test design activities. TAF has been demonstrated that it can be integrated into existing approach for reducing cost and schedule saving.

Despite the manual requirements inspections and requirements defects detection, Lami et al. [19] presented a methodology and a tool named QuARS, Quality Analyzer for Requirement Specifications, to analyze and validate natural language requirements in a automated systematic way. Requirements engineers are allowed to perform an initial parsing of the requirements with purpose to detect potential linguistic defects in requirements automatically by adopting QuARS tool. In addition, QuARS tool also supports the consistency and completeness analysis of requirements automatically via requirements clustering in accordance to a specific topic. QuARS tools performed analysis based on the corresponding dictionary that comprised of a predefined precise set of terms and linguistic constructions. Dictionaries are considered the passive component in QuARS tool due to the variety application domain and user needs. However, the effectiveness of QuARS tools is strictly depends on the accuracy, completeness and adequacy-to-domain of the dictionary mentioned above. QuARS tool is only limited to analyze syntax related issues from a natural language requirements documents only.

An approach was introduced in 2009 named as R-Tool, automation of use case driven requirement analysis, with aim to support the analysis stage of software development in an object oriented framework [20]. In the environment of object oriented, the goal of the R-Tool is to understand the domain of the problem and the responsibilities of the system. The object oriented analysis aids in determining the system requirements via identification of the classes and their relationship to the classes in the problem domain which differ from the others current tools. The R-Tool natural language processing based CASE tool takes requirements as the input and produce the elements of object oriented system for example classes, attributes, methods and etc. The generation of the class diagram is treated as output of the R-Tool. In their initial experiment of adopting the R-Tool, it is found to supplement the manual approach in identifying the inconsistencies between manual approach and automated approach in order to identify the system requirements properly. One of the constraints of the R-Tool is the complexity of the requirements as input. The requirements input must be split into two simple sentences if it is a complex sentence with the existence of conjunction.

The Goddard Space Flight Center’s (GSFC) Software Assurance Technology Center (SATC) developed a tool for assessing requirements in natural language during the early life cycle [21]. The tool, Automated Requirements Measurements (ARM) searches the occurrence of each of the quality primitives in requirements document defined by SATC. Eight primitives were implied in the quality attributes including complete, correct, ranked, unambiguous, consistent, modifiable, traceable and verifiable. Based on the subject study conducted, three initial conclusions have been drawn. Firstly, it is beneficial to the quality of the requirements specifications adopting ARM via the usage of data gathered by automated processing of the requirements specification file. Besides, the relatively simple and readily of ARM improves greatly the effectiveness of expressing the requirements specifications in natural language. Lastly, requirements specifications developed using a proven methodology are found to be better in structure, more consistent in number and contain crisper specification statements in comparison to those requirements specification developed based on a common documentation standard.

IV. Comparative Analysis

This section summarizes and synthesizes the findings in our literature review. We have categorized the existing approaches to handle requirements defects into four different categories from our conducted literature review above. The four different categories refer to reading technique, inspection technique, analysis technique and automated tool. Table 2 illustrates the categorization of the findings.

Table 2
Comparative Analysis of Current Techniques in Handling Requirements Defects

Techniques to Handle Requirements Defects	
1.0 Reading Technique	[9][10][11][12][13]
1.1 Systematic Reading Technique	
1.2 Non-systematic Reading Technique	
2.0 Inspection Technique	[14]
2.1 N-fold Inspection Technique	[2][15]
3.0 Analysis Technique	[18]
3.1 Software Fault Tree Analysis (SFTA)	
4.0 Automated Tool	[19][20][21]

Based on Table 2, we found that the existing techniques for requirements defects detection are mainly adopting manual approaches. The implementation of automation tool in requirements defects detection in requirements analysis is scarce according to our conducted review.

Despite of these research advancements, empirical evidence suggests that quality is still an issue because developers lack of understanding on the source of problems, inability to learn from the previous mistakes, lack of effective tools and there is none complete verification process found yet [4][16]. This study provides recommendation to the industry and researches on the existing techniques in handling requirements defects in order to propose a better approach or automated tool.

V. Conclusion

In this paper, we presented a comparative study on the existing techniques to handle requirements defects including reading techniques, inspection technique, analysis technique or automation tool technique. There is a need to consider the needs from the industrial upon handling requirements defects. The tool in the future should be evolved with respect to the needs from the industrial and not only theoretical with aim to improve the quality of the requirements by minimizing requirements defects in requirements engineering activities. A future work of this comparative study on requirements defects proposes for a framework or tool that improved execution of requirements analysis activity.

Acknowledgment

I would like to thank UTeM, zamalah UTeM and FRGS grant for the funding research: FRGS/1/2015/ICT01/FTMK/02/ F00291.

References

[1] O. L. Lee, M. Kamalrudin, and S. Sidek, “Pair in software requirements engineering: A review,” Int. J. Appl. Eng. Res., vol. 10, no. 14, pp. 34238–34243, 2015.

[2] G. Michael, “An Experimental Study of Fault Detection User Requirements Documents,” no. 2, pp. 188–204, 1992.

[3] IEEE Standard Classification for Software Anomalies. 2010.

[4] M. R. Blackburn, R. Busser, and A. Nauman, “Removing Requirement Defects and Automating Test,” 2001.

[5] S. Lauesen and O. Vinter, “Preventing Requirement Defects,” pp. 1–10, 2000.

[6] J. Chen and S. Huang, “The Journal of Systems and Software An empirical analysis of the impact of software development problem factors on software maintainability,” vol. 82, pp. 981–992, 2009.

[7] S. Ahmad, S. A. Asmai, and N. A. Rosmadi, “A Significant Study of Determining Software Requirements Defects : A Survey,” WSEAS Press, pp. 117–122, 2015.

[8] I. L. Margarido, J. P. Faria, and M. Vieira, “Classification of Defect Types in Requirements Specifications : Literature Review , Proposal and Assessment,” no. 1, 2009.

[9] B. Brykczynski and D. A. Wheeler, "Software Inspection : Eliminating Software Defects," 1994.

[10] A. Aybuke;, P. Hakan;, and W. Claes, "State-of-the-Art : Software Inspections after 25 Years," vol. 12, no. 3, pp. 133–154, 2002.

[11] A. A. Porter, L. G. Votta, and C. Park, "An Experiment To Assess Different Defect Detection Methods For Software Requirements Inspections," 1994.

[12] A. A. Porter, L. G. Votta, and V. R. Basili, "Comparing Detection Methods For Software Requirements Inspections : A Replicated Experiment," 1998.

[13] F. Shull, I. Rus, and V. Basili, "How Perspective-Based Reading Can Improve Requirements Insoections," no. July, 2000.

[14] M. E. Fagan, "Advances in Software Inspections," no. 7, pp. 744–751, 1986.

[15] J. Martin; and W. T. Tsai, "N-Fold inspection: a requirements analysis technique," Commun. ACM 33, pp. 225–232, 1990.

[16] G. S. Walia and J. C. Carver, "A systematic literature review to identify and classify software requirement errors," Inf. Softw. Technol., vol. 51, no. 7, pp. 1087–1109, 2009.

[17] M. Towhidnejad, D. R. Wallace, A. M. Gallo, N. Goddard, and S. Flight, "Fault Tree Analysis for Software Design," 2003.

[18] G. Helmer, J. Wong, M. Slagell, V. Honavar, L. Miller, and R. Lutz, "A Software Fault Tree Approach to Requirements Analysis of an Intrusion Detection System," no. 2, 2002.

[19] G. Lami, S. Gnesi, F. Fabbrini, M. Fusani, and G. Trentanni, "An Automatic Tool for the Analysis of Natural Language Requirements," 2005.

[20] S. Vinay, S. Aithal, and P. Desai, "An Approach towards Automation of Requirements Analysis," vol. I, 2009.

[21] W. M. Wilson, L. H. Rosenberg, and L. E. Hyatt, "Automated Analysis of Requirement Specifications," pp. 1–11, 2005.

[22] G. O. Young, "Synthetic structure of industrial plastics (Book style with paper title and editor)," in Plastics, 2nd ed. vol. 3, J. Peters, Ed. New York: McGraw-Hill, 1964, pp. 15–64.

[23] W.-K. Chen, Linear Networks and Systems (Book style).Belmont, CA: Wadsworth, 1993, pp. 123–135.

[24] H. Poor, An Introduction to Signal Detection and Estimation. New York: Springer-Verlag, 1985, ch. 4.

[25] B. Smith, "An approach to graphs of linear forms (Unpublished work style)," unpublished.

[26] E. H. Miller, "A note on reflector arrays (Periodical style—Accepted for publication)," IEEE Trans. Antennas Propagat., to be published.

[27] J. Wang, "Fundamentals of erbium-doped fiber amplifiers arrays (Periodical style—Submitted for publication)," IEEE J. Quantum Electron., submitted for publication.

[28] C. J. Kaufman, Rocky Mountain Research Lab., Boulder, CO, private communication, May 1995.

[29] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interfaces (Translation Journals style)," IEEE Transl. J. Magn.Jpn., vol. 2, Aug. 1987, pp. 740–741 [Dig. 9th Annu. Conf. Magnetics Japan, 1982, p. 301].

[30] M. Young, The Techincal Writers Handbook. Mill Valley, CA: University Science, 1989.

[31] J. U. Duncombe, "Infrared navigation—Part I: An assessment of feasibility (Periodical style)," IEEE Trans. Electron Devices, vol. ED-11, pp. 34–39, Jan. 1959.

[32] S. Chen, B. Mulgrew, and P. M. Grant, "A clustering technique for digital communications channel equalization using radial basis function networks," IEEE Trans. Neural Networks, vol. 4, pp. 570–578, Jul. 1993.

[33] R. W. Lucky, "Automatic equalization for digital communication," Bell Syst. Tech. J., vol. 44, no. 4, pp. 547–588, Apr. 1965.

[34] S. P. Bingulac, "On the compatibility of adaptive controllers (Published Conference Proceedings style)," in Proc. 4th Annu. Allerton Conf. Circuits and Systems Theory, New York, 1994, pp. 8–16.

[35] G. R. Faulhaber, "Design of service systems with priority reservation," in Conf. Rec. 1995 IEEE Int. Conf. Communications, pp. 3–8.

[36] W. D. Doyle, "Magnetization reversal in films with biaxial anisotropy," in 1987 Proc. INTERMAG Conf., pp. 2.2-1–2.2-6.