IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 4, April 2016

Software Testability in Requirement Phase: A Review

Mohammad Zunnun Khan[1], M.Akheela Khanam[2], M.H.Khan[3]

Research Scholar, Computer Science & Engineering, Integral University, Lucknow, India [1]

Associate Professor, Computer Science & Engineering, Integral University, Lucknow, India [2]

Professor, Department, Computer Science & Engineering, I.E.T., Lucknow, India [3]

Abstract: Testability can be used as a high impact quality indicator in the modern era of software development process. The complete process of testability always helps the developer by its correct measurement or evaluation. But its correct evaluation is not an easy task for the practitioners. Practitioners as well as researchers have always suggested that testability should be considered as a primary attribute towards achievement of quality software process. Software quality's accurate measure depends on testability measurement, and as an outcome estimating efforts in measuring testability is a complex problem that requires considerable attention of researchers. Primary objective of this review report is to raise the testability issues with the limitation and to investigate the general testability factor and minimal set of commonly accepted testability factors, and proposing a conceptual comparative evolution. Here we review the literature to gain wide knowledge of testability and its quality factors and measurement presented by various researchers in different perspective.

Keywords: Software Development, Software Testability, Software Requirement, Software Quality

I. INTRODUCTION

Professionals of Software development life cycle has broadly focus on minimizing errors, detecting and correcting software faults that occurs during development life cycle, and try to deliver a high quality software after the software development process[24]. But only delivering high quality software is no longer just an advantage but it is also a necessary factor. But most of the industries are not able to deliver a high quality product to their stakeholders, and also do not understand quality attributes those are relevant [18, 2]. Software testing is the main activity in complete process of development it is also true that it consumes major proportion of time and effort. There is a need of an approach which performs testing accurately. Software testability always supports the testing process and facilitates the development of highly quality software within time and budget [1].

If there is an effective testability plan for the development process is possible at an early stage i.e., requirement phase. May delivery a high quality software product and satisfy users [3]. It will reduce the overall maintenance cost and rework. However, less effective testability plan or later stage testability plan of development process will lead to unsatisfied users, low quality product, unreliability and inaccuracy towards results [4].

II. SOFTWARE TESTABILITY

Various definitions of testability are available. Common and most effective is the ease of performing testing [23]. The above mentioned definition has its roots in hardware testing which is usually expressed by the two terms one observability [22] and another is controllability.

Binder has defined these factors of testability as [10], if you are able to control its input state, i.e. internal state and observer the output state, to test a component. And if you are not able to control input, you cannot be so sure on output. Based on the given definitions, it is intuitive how controllability and observability ease of testing and reduce its cost. If controllability is missing redundant tests will produce different results, and in absence of observability, incorrect results may appear correct as the error is contained in an output that you are unable to see [7, 17].

Voas mentioned that only controllability and observability cannot represent all the cost of testing yet they are part of the testability [9, 12]. A primary component is the ability to revel faults of testing. Testing has least value if a testing activity fails to identify the problem that exists. This value definition testability tries to measure the accurate amount of effort necessary to adequately test the component or complete system such that all faults are traced.[6,8] Our research in software testability at an early stage has yield evidence that there is a correlation between low testability and object oriented approach (in requirement phase).

III.SOFTWARE REQUIREMENT

Traditionally, software requirements are either functional or non-functional with hidden notation of quality in the latter stage. As the focus of industry professional is

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 4, April 2016

shifting from functionality to improving quality [11, 15], a new type of requirements focused on quality is emerging. In order to specify these new quality requirements, quality itself must be defined. A quality model provides the framework towards a definition of quality. Practitioner has long recognised that in order for something to find its way in a final product, it is must to define and specify it properly. Regrettably, the software quality that can be observed in the industry today is missing a solid foundation in the form of an agreed upon the quality model that can evaluate and specify the quality of software.

Bourque (2000) has advised that the implementation of quality software products is a series of action in formal manner should be managed through the software engineering life cycle [25]. The implementation of quality software should therefore begin with specifying the quality requirement of users. Suryn (2003) has advised that this domain as combination of an uninterrupted, systematic, disciplined and quantifiable approach for the development and maintenance of quality software product & system [21].

IV. TESTABLE REQUIREMENT

The main measure of success of software is the degree to which it meets the purpose for which it was built. The requirement engineering is the way of discovering that purpose, by identifying needs of stakeholders and documenting these in a form that is amenable to analysis, communication, and able for subsequent implementation [31]. It is decision of developer that how to implement the given requirement. According to a high quality requirements document must satisfy these quality parameters [30].

- Correct— a requirement statement must accurately describe the functionality to be delivered product and customer is the most superior authority to determine the correctness of the requirement.
- Unambiguous— a requirement statement should be able to draw only one meaning to the reader.
- Complete— a requirement statement must contain all the necessary information and convey complete.
- Consistent—a requirement statement should not conflict with any other requirements. Any Disagreements among requirements must be resolved before the development process starts.
- Verifiable—a requirement statement that is unambiguous is verifiable. And the person who uses it must be able to determine if the requirement statement have been met.
- Ranked for importance and stability- a requirement statement must have an implementation priority according to its importance.
- Modifiable- a requirement statement that is modifiable must be cross referenced and uniquely ladled, so it can be changed without any difficulty.
- Traceable- a requirement statement that is traceable, it should be possible to trace each requirement to its source.
- Understandable- a requirement statement should be grammatically correct and written in a consistent style. Standard conventions should be used.

A Testable Requirement is a consistent, unambiguous description of the expected system behaviour that is verifiable.

V. RELATED WORK

Software industry calls for a formal management of quality throughout the lifecycle [2]. To achieve this requirement, a quality model should support the definition of quality requirement and its subsequent evaluation. This can be expressed by referring to the manufacturing view of quality, which express that quality is conformity to requirements. A base quality model that used for the definition of quality requirements will definitely help in both the specification of quality requirement and the evaluate software quality.

IEEE Std 1061-1998 [28], defines this as a top-down and bottom-up approach to quality: its top-down view suggested the framework that establish the quality requirements factors for the users and managers early in software development lifecycle, communication of well established quality factors, in form of quality sub factors to the professionals and identify the metrics that are related to established quality factors and sub factors.

And its bottom-up view suggested the framework that enables the managerial and technical professionals to obtain feedback by evaluating the software product and processes, at the metrics level & analysing metric values to assess and estimate the quality factors. A quality model that can be used as the structure for the definition of quality requirements should help the industry professional in both specification of quality requirement and evaluation of software quality [27, 29]. Or we can say that it should be usable from top of development process to bottom and it's vice versa.

Three parameters that a quality model must possess to be a foundation for software development lifecycle have been identified; a quality model must support five perspective of quality as defined by Kitchenham and Pfleeger [19, 25], a quality model must be as much as usable from top to down of the lifecycle according to IEEE Std 1061-1998 [30],i.e. it should allow for defining quality requirements and its further decomposition into quality characteristics, sub characteristics, a quality model must be usable from bottom to up of the lifecycle as defined by IEEE,1998, i.e. should allow for required measurement and aggregation and evaluation.

As Davis (1993) states and illustrates that a set of requirements is correct when each and every requirement stated in it represents something in the derived system. If

the universe of user needs is represented by the circle on the left and the requirements by the circle on the right, the portion of correct requirements is area B, the area of overlap. Of course, by simply writing some information in a document, anyone can not guarantee that it is correct and can any automated design tool provide a guarantee that it will be correct.

▪ REQUIREMENT SPECIFICATION AND EVALUATION OF QUALITY

Software industry calls for a formal management of quality throughout the lifecycle [2]. To achieve this requirement, a quality model should support the definition of quality requirement and its subsequent evaluation. This can be expressed by referring to the manufacturing view of quality, which express that quality is conformity to requirements. A base quality model that used for the definition of quality requirements will definitely help in both the specification of quality requirement and the evaluate software quality.

IEEE Std 1061-1998 [28], defines this as a top-down and bottom-up approach to quality: its top-down view suggested the framework that establish the quality requirements factors for the users and managers early in software development lifecycle, communication of well established quality factors, in form of quality sub factors to the professionals and identify the metrics that are related to established quality factors and sub factors. And its bottom-up view suggested the framework that enables the managerial and technical professionals to obtain feedback by evaluating the software product and processes, at the metrics level & analysing metric values to assess and estimate the quality factors. A quality model that can be used as the structure for the definition of quality requirements should help the industry professional in both specification of quality requirement and evaluation of software quality [27, 29]. Or we can say that it should be usable from top of development process to bottom and it's vice versa.

Three parameters that a quality model must possess to be a foundation for software development lifecycle have been identified; a quality model must support five perspective of quality as defined by Kitchenham and Pfleeger [19, 25], a quality model must be as much as usable from top to down of the lifecycle according to IEEE Std 1061-1998 [30], i.e. it should allow for defining quality requirements and its further decomposition into quality characteristics, sub characteristics, a quality model must be usable from bottom to up of the lifecycle as defined by IEEE,1998, i.e. should allow for required measurement and aggregation and evaluation.

As Davis (1993) states and illustrates that a set of requirements is correct when each and every requirement stated in it represents something in the derived system. If the universe of user needs is represented by the circle on the left and the requirements by the circle on the right, the portion of correct requirements is area B, the area of overlap. Of course, by simply writing some information in a document, anyone can not guarantee that it is correct and can any automated design tool provide a guarantee that it will be correct. If the user's true requirements in a shopping system are that on item A has 5 percent discount and item B has 8 percent discount but the project team inadvertently creates a requirement stipulating a 8 percent discount on item and A and 5 percent on item B, so it is sure that it is not correct. This form of correctness will be verified only by review and acceptance by the stakeholders.

As IEEE 830-1993-1994[26] stated that a set of requirements is unambiguous when it can be interpreted only in one aspect. Although correctness of requirement is obviously a key concern, sometimes ambiguity can turns out to be a big problem. If a statement of requirements can be interpreted differently by developers, users, and other stakeholders in the project, it's quite possible to build a system that is completely different from what the user had in mind. This is because of , an insidious problem whenever requirements are written in natural language, as well as because different cultural groups within an organization are so accustomed to their interpretation of a word or phrase that it never occurs to them that others might interpret the word differently.

A set of requirements is complete if the statement describes all significant requirements of concern to the user, including those requirements which are associated with functionality, performance, design constraints, attributes, or any other external interfaces [26]. A complete set of requirements must also define the required response of the software to all realizable classes of inputs including valid and invalid situations of all realized classes. So, it must provide complete references and labels for all of the figures, tables, and diagrams within the requirement set.

Ensuring Completeness some aspects of completeness can be judged by experienced reviewer who critically assesses the requirements that ensures the figures, labels, and diagrams have proper references and labels. Also, some aspects of completeness can be assessed even by a developer with no special understanding of the application. A requirement set is internally consistent if and only if no subsets of individual requirements described within it are in conflict with one another [26].

The conflicts can take various forms and are visible at various levels of detail, if the set has been written in a reasonably formal fashion and if it is supported with appropriate automated tools, the conflicts can sometimes be identified through a mechanical analysis. [5]

A requirement is verifiable in the aggregate if each component requirements contained within it is verifiable. And the requirements can be deemed verifiable if and only if there exist a finite, cost-effective process with which a person or a machine can determine that the developed

software system meets the requirement [26]. In short, we realize, as a professional that it is necessary to define requirements so that we can later test them and determine whether they were achieved. .

McCall (McCall et.al, 1977) [13] firstly introduced a quality model in the year 1977. Pfleeger et.al, 2001, has also stated it as a first model and shows the mapping of those quality factors that are not directly measurable on one side and on other end measurable properties on the basis of subjective grading scheme [16]. Regarding this model, pressman, 2001 has noted that unfortunately, most of the properties measured by McCall et.al. , are only measured subjectively, so it is difficult to use this framework for specifying quality requirements, as traceability and self documentation are not meaning full at an early stage for nontechnical users. And will not fulfil the criteria of IEEE standard for software quality metrics.

Boehm [14] improves the work of McCall and proposed a model; this model loosely retained the measurable property arrangement. According to Boehm the prime quality characteristic is what they defined as general utility. They considered the maintainability, utility and portability useful for the system. This characteristics General Utility and as-is Utility are too generic to be useful for defining requirements which can be verifiable. Like the McCall model, this model is mostly useful for a bottom-up approach to quality of software i.e. it can effectively be used to define measures of software quality. While this model is a step forward in the sense that it provides basic support for a top-down approach to software quality, this support is too short to be considered as a solid base for quality engineering.

Dormey [15], 1995 model has selected a new path towards software quality then existing ones (McCall and Bohem). This quality model was based on the product quality perspective, shows that what must be recognized in a quality model. In place of it exhibits product characteristic that contribute to quality attributes and other characteristics that can detract from the quality attributes of a product. Most models of software quality fail to deal with the product characteristics side of the problem adequately and it also fails to make the direct links between quality attributes and corresponding characteristics of product. He has suggested an evaluation based quality framework that is able to analyze the quality of software components through the measurement of quality properties that are Capable of being perceived. Each and every end produced in the software development lifecycle can be associated with an evaluation model based on quality.

Dromey's work was significant for both technical team and stakeholders, but it was still a difficult task to implement, how it could be used at an early stage of development process. So it fails to qualify as a foundation for Software Quality Engineering according to the established requirements. In the year 1991, the ISO has introduced as standard named ISO/IEC 9126[20]; software product evaluation, quality based characteristics and set of rules for their best use. It also targeted to define a quality model for software product and a set of specific guidelines for the measurement of associated characteristics. ISO/IEC-9126 [20] was very popular in Europe as the best way to measure quality of software product as Bazzana et.al. 1993 stated it. But there were some problems such as it has no specific guidelines on how to provide a quality assessment and no indication on how to perform the measurement of quality; it only reflects the consumer view as stated by Pfleeger, 2001 [21] .

FRUPS model was firstly presented by Grady [32], and then it is extended by IBM Rational Software [33 and 34] into FURPS+. The "+'' symbol indicates such requirement set as design constraints, implementation of requirements, interface of requirements and physical requirements [33].

On these four characteristics FURPS model is defined as follows:

- Functionality – This includes feature sets, capabilities and security.
- Reliability – This includes frequency and severity of failure, recoverability, predictability, accuracy, and mean time between failures (MTBF).
- Usability – This includes human factors, aesthetics, consistency in the user interface, online and context sensitive help, wizards and agents, user documentation, and training materials.
- Performance – This imposes conditions on functional requirements such as speed, efficiency, availability, accuracy, throughput, response time, recovery time, and resource usage.
- Supportability – this includes testability, extensibility, adaptability, maintainability, compatibility, configurability, serviceability, installability, localizability.

It can be categories in two different types: Functional (F) and Non-functional (URPS) [17]. These categories also help us as both product requirements as well as in the assessment of product quality.

VI. CRITICAL OBSERVATIONS

After revisiting the approaches of researches some important observation can be specify as follows:

- If we improve quality of software at an early stage that is requirement phase then the overall process may greatly improve. It will improve client satisfaction and reduce cost and effort.
- This study provides that requirement errors are the most common category of systems development errors. If we control error at the requirement phase, the delivered product will surely be improved.

- Testability is a quality factor which is commonly accepted by various quality models.
- As suggested by various researchers defect potential is high as well as efficiency to remove effort is also very significant in the requirement phase of software development lifecycle.

If we are able to achieve testable requirement then we can avoid redesign, recode and retest at later stages of development.

VII. CONCLUSION

Various frameworks have been proposed in the literature for measuring software testability. A survey of the relevant literature shows that maximum efforts have been put at the later stage of software development life cycle. If requirement based errors can be fixed quickly, easily, and economically, project in later stages of development may not have a huge problem. Although the studies of this review, all reached roughly the same conclusion: If a unit cost of one is assigned to the effort required to detect and repair an error during the coding stage, then the cost to detect and repair an error during the requirements stage is between five to ten times less. And the cost to detect and repair an error during the maintenance stage is twenty times more. At last study can conclude that testability is a quality factor that attempts to predict how much effort will be required for software testing process.

REFERENCES

[1] S. Mouchawrab et al, "A measurement framework for object-oriented software testability," Carleton University, Technical Report, SCE-05-05, year 2005.
[2] Testability Estimation Model: M. Nazir & R.A.Khan, Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, Vol. 85, Meghanathan, Natarajan; Chaki, Nabendu; Nagamalai, Dhinaharan (Eds.), Volume 85, Part 3, LNICST, Springer-Verlag, 2012, pp 178-187, (ISBN 978-3-642-27307-0). January 2012.
[2] Nazir M., Khan Raees. A.," An Empirical Validation of Understandability Quantification Model", Journal Procedia Technology, 2nd International Conference on Computer, Communication, Control and Information Technology, Volume 4, Pages 772–777, 2012.
[3] Improving the Testability of Object-oriented Software during Testing and Debugging Processes, Sujata Khatri, R.S. Chhillar, V.B.Singh, International Journal of Computer Applications (0975 – 8887) Volume 35– No.11, December 2011.
[4] A. Zaidman et. al, "On how developers test open source software systems", Technical Report TUD-SERG-2007- 012, Delft University of Technology, Software Engineering Research Group, 2007.
[5] Abdullah, Dr. Reena Srivastava, Dr. M.H. Khan International Journal of Advanced Information Science and Technology (IJAIST) ISSN: 2319:2682 Vol.26, No.26, June 2014
[6] L. Zhao, "A new approach for software testability analysis", International Conference on Software Engineering, Proceeding of the 28th international conference on Software Engineering, Shanghai, pp. 985–988, 2006.
[7] Voas and Miller, "Software Testability: The New Verification". IEEE Software, Vol. 12(3), p. 17-28, 1995.
[8] J.M. Voas, "Object-Oriented Software Testability", In proceedings of International Conference on Achieving Quality in Software, January 1996.
[9] R.V. Binder, "Design for testability in object-oriented systems", Communications of the ACM Vol. 37(9), p. 87-101, 1994.
[10] M. Nazir, Khan R. A. & Mustafa K. (2010): Testability Estimation Framework, International Journal of Computer Application, Vol. 2, No. 5, pp.9-14. June 2010.
[11] Voas and Miller, Semantic metrics for software testability, Journal of Systems and Software, Vol. 20 (3), pp. 207-216, 1993.
[12] McCall JA, Richards PK, Walters GF. Factors in software quality, RADC TR-77-369: 1977. (Rome: Rome Air Development Centre)
[13] Boehm BW, Brow JR, Lipow M, McLeod G, Merritt M. Characteristics of software quality, North Holland Publishing, Amsterdam, the Netherlands; 1978.
[14] Dromey RG. Concerning the Chimera (software quality). IEEE Software. 1996; 1:33.
[15] Drown DJ , Khoshgoftaar TM, Seiya N. Evaluation any sampling and software quality model of high assurance systems, IEEE Transaction on systems, Mean and Cybernetics, Part A: Systems and Human. 2009;39(5):1097-1107.
[16] Huda, M., Arya, Y.D.S. and Khan, M.H. (2015) Evaluating Effectiveness Factor of Object Oriented Design: A Testability Perspective. International Journal of Software Engineering & Applications (IJSEA), 6, 41-49. http://dx.doi.org/10.5121/ijsea.2015.6104
[17] Huda, M., Arya, Y.D.S. and Khan, M.H. (2015) Testability Quantification Framework of Object Oriented Software: A New Perspective. International Journal of Advanced Research in Computer and Communication Engineering, 4, 298- 302. http://dx.doi.org/10.17148/IJARCCE.2015.4168
[18] Krruchtem P. The rational unified process: an introduction, Addison Wesley; 2000.
[19] ISO /IEC25010: Software engineering– system and software quality requirement and evaluation (SQuaRE)- system and software quality model; 2011.
[20] Esaki K. System quality requirement and evaluation, importance of application of the ISO/IEC 25000 series, Global Perspectives of Engineering Management.2013; 2(2):52-59.
[21] Huda, M., Arya, Y.D.S. and Khan, M.H. (2015) Metric Based Testability Estimation Model for Object Oriented Design: Quality Perspective. Journal of Software Engineering and Applications, 8, 234-243. http://dx.doi.org/10.4236/jsea.2015.84024
[22] Robert V. Binder. Testing object-oriented systems: models, patterns, and tools. Addison-Wesley Longman Publishing Co., Inc., 1999.
[23] Huda, M., Arya, Y.D.S. and Khan, M.H. (2014) Measuring Testability of Object Oriented Design: A Systematic Review. International Journal of Scientific Engineering and Technology (IJSET), 3, 1313-1319.
[24] Krruchtem P. The rational unified process: an introduction, Addison Wesley; 2000.
[25] Addison Wesley - Leffingwell & Widrig-Managing Software Requirements, 1St Edition.
[26] IEEE Computer Society. IEEE Standard 8301998: IEEE Recommended Practice for Software Requirements Specifications. New York City: IEEE.
[27] S. Wright. Requirements traceability - what? why? and how? In Proc. IEE Colloquium on "Tools and Techniques for Maintaining Traceability During Design", pages 1–2. IEEE, 1991
[28] IEEE Standard 1233a1998: IEEE Guide for Developing System Requirements Specifications. New York City: IEEE
[29] Spence, I. and Probasco, L. (2000), Traceability strategies for managing requirements with Use cases, Rational Software White paper.
[30] Huda, M., Arya, Y.D.S. and Khan, M.H. (2015) Quantifying Reusability of Object Oriented Design: A Testability Perspective. Journal of Software Engineering and Applications, 8, 175-183. http://dx.doi.org/10.4236/jsea.2015.84018
[31] Grady R B., Practical software metrics for project management and process improvement, Prentice Hall; 1992.
[32] Jacobson I, Booch G, Rumbaugh J. The unified software development process, Addison Wesley; 1999.