Software Quality Evaluation through Maintenance Processes

Anas Bassam AL-Badareen, Zaiton Muda, Marzanah A. Jabar, Jamilah Din, Sherzod Turaev

***Abstract*—** The software is always required to be developed and maintained a quality to the rapid progresses in industry, technology, economy, and other fields. Thus, software maintainability is the major activity in the software life cycle. This process bypass the cost required to develop new software product. However, software maintenance costs often total twice the original development cost in the lifetime of application software. Therefore, software maintainability is considered as one of the most important quality factor in software quality models. This study introduces a software maintenance process model that emphasizes the impact of the software quality on the maintenance and considers the quality of the modified software.

***Keywords*—** Maintenance Evaluation, Maintenance Framework, Software Quality, Maintenance Process.

I. Introduction

In order to keep the software useful and dynamic with the world changes, the software has to be changed accordingly [1]. As the information technology industry gains maturity, the number of software systems having moved into maintenance is rapidly growing [2]. Software maintenance is a hard (and costly and error-prone) process in software life cycle [3]. Therefore, the cost of software maintenance rapidly increases. Souza [1] and Burch and Hsiang-Jui [4] show that the cost estimated for software maintenance is very high (80-90%) of the total cost of Software Development Life Cycle (SDLC). Therefore, software maintenance is recognized as an economic impact in the information system community.

This study aims to analyze the maintainability characteristics, classify them, and then discuss their affects on the maintenance process. Finally, the study comes out with the factors that need to be considered in order to calculate the maintainability factor.

Manuscript received September 30, 2010. This work was supported in part by Fundamental Research Grant, Ministry of Higher Education Malaysia, Project number: 03-04-10-867FR.

Anas Bassam AL-Badareen is with the University Putra Malaysia, 43400 UPM, Serdang, Selangor, Malaysia. Phone: 601-72301530; e-mail: anas_badareen@hotmail.com.

Zaiton Muda is with University Putra Malaysia, 43400 UPM Serdang Selangor, Malaysia; e-mail: zaiton@fsktm.upm.edu.my.

Marzanah A. Jabar PhD is with University Putra Malaysia, 43400 UPM Serdang Selangor, Malaysia; e-mail: marzanah@fsktm.upm.edu.my.

Jamilah Din PhD is with University Putra Malaysia, 43400 UPM Serdang Selangor, Malaysia; e-mail: jamilah@fsktm.upm.edu.my.

Sherzod Turaev is a Postdoctoral Researcher with University Putra Malaysia, 43400 UPM, Serdang Selangor, Malaysia; e-mail: sherzod@fsktm.upm.edu.my.

II. Software Maintenance

Traditionally, software maintenance is defined as any modification made on a system after its delivery [1]. IEEE defines the software maintainability as the modification of a-software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment. Software maintenance broadly includes error corrections, changes (amendments or enhancements) and improvements to operational software [2].

Tiako [5] presents the reasons of software maintenance as (a) the changes at the level of software requirements; (b) the changes at the level of functional specification and design of the software system; (c) the changes that interfere at the level of performance specification; (d) the changes at the level of the system's environment; (e) the historic changes of the software implementation; (f) the disparity between the specification and the implementation of the software; and (g) the changes of strategic needs.

III. Software Maintainability

Since there is no clear definition of measuring software maintainability, different types of measures for the maintainability have been discussed. Maintainability can be measured in terms of which the software can be corrected, adapted or perfected. Also it can be measured in terms of time oriented metric, such as Mean Time to Change (MTTC), a cost oriented metric such as Spoilage, or Software Maturity Index. Generally, the maintainability of the software is the cost required to modify and correct the errors in the delivered software product without damage the primary functions.

The software maintenance is important, because it bypass the developers the cost of build of new system in order to replace the existing [6-7]. Mittal [8] expressed four software aspects required to assess the software maintainability. Moreover, the ability of the software to be diagnosed for deficiencies or cause of failures in the software is added in ISO, which is added to the definition the ability of the system to specify the failures.

Hence, software maintenance includes four major stages: understanding, analyzing, modifying, and testing. Singh [9] analyzed the major factors that can affect software maintenance and divided them into four categories: Readability of source code, Documentation Quality,

Understandability of Software, and Average Cyclomatic Complexity. A model proposed in [10], considers maintainability as integrated measure of three factors, Readability of Source Code (RSC), Documentation Quality (DQ), and Understandability of Software (UOS).

IV. Software Maintainability Factors

In order to evaluate the quality of software maintainability, the characteristics are classified according to the processes: understanding, analyzing, modifying, and testing. The characteristics for each process are derived and concluded from the most well-known software quality models and literature.

A. Understandability

One of the major concerns of any maintenance organization is to understand and estimate the cost of maintenance releases of software systems [11]. The major professional challenge is to understand and conquer the complexity of the system as it is. System understanding is the major point in the maintenance stage, which is concerned about the ability of the developer to understand the functions and their relationships within the system. This allows them to identify the errors or the parts of the system that require the modification.

Software understandability is a very important quality factor that allows the developers to understand the structure of the system easily, which simplifies the task of software maintenance [12]. Therefore, system understanding depends on cognitive abilities and preferences, familiarity with the application domain, and the set of support facilities provided by the software engineering environment. Thus, the understandability affects the effort and the cost required in software maintenance.

Therefore, software understanding aims to present enough information about the system. This information is written by software engineers and programmers during software development lifecycle [13]. If the documentation and the source code are not correlated, the maintenance process will be very difficult and not accurate [10]. In other words, the understandability will be high if and only if the source code and the documentation are closely related.

In order to understand a system, several characteristics are considered. Figure 1 shows the characteristics concerned to the simplicity of the system and the information used to express the software functions and their relations.

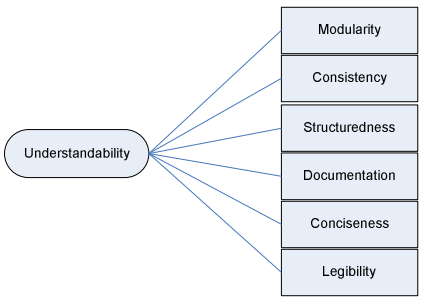

Figure 1: Understandability Factors

Modularity is an available method to solve a complex problem, which aims to decompose and integrate all objects. Software modularity concerns about the decomposition of the system into several manageable components allowing understanding the system part by part instead of understanding the system. Therefore, it is considered as a key way of innovation and mass customization.

The consistency is important for the correct interpretation of the system components, which is defined as no contradiction in the system. It covers the consistency among the components of sub-system and among the sub-systems themselves. Furthermore, the system consistency can be achieved by covering the consistency in the software development phase and between different phases of software development.

Software structuredness represents the degree at which the SDD (Software Design Description) and code possess a definite pattern in their interdependent parts. This implies that the design has been preceded in an orderly and systematic manner, has minimized coupling between modules, and that standards control structures have been followed during coding resulting in well structured software.

The documentation of the system aimed to offer comprehensive, clear and short information about the system. This factor concerned to describe the system functions and the relationships within the system. Software documentation responds to three necessities: (i) contractual; (ii) support a software development project allowing team members to gradually conceive the solution to be implemented, and (iii) allow a software development team to communicate implementation details across time to the maintenance team.

The documentation of the system is important to improve software development and to help in the maintenance process [1], which is required to understand the certain level of the software abstraction in shorter time. Furthermore, the quality of the software documentation affects directly the system understanding. Therefore, the documentation should be meaningful for the developers, available at each level of the system abstraction, smoothly move between these levels without losing their position in the documentation, and consistent with source code.

Conciseness means a system provides only the information necessary to complete the task. It is a program's compactness in terms of lines of code.

Legibility of the system is similar to clarity which defines that the information should be easy to read.

B. Analyzability

In order to identify the errors or the parts required to modify, the system analysis is needed. The analyzability specifies the ability of the system to be diagnosed for deficiencies, causes of failures in the software, or identification of the parts required modifications which may be in primary concern [14-15]. Figure 2 presents the characteristics that affect analyzing software.

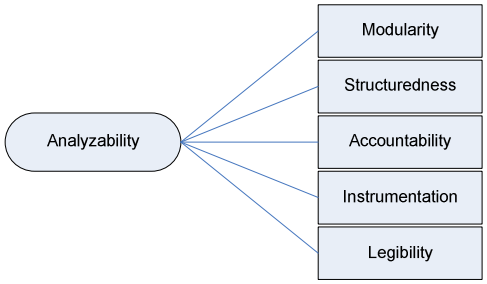

Figure 2: Analyzability Factors

The modularity simplifies the software analysis by decomposing the system into several manageable parts, which allow identifying the errors or the functions that required the modification easily and efficiently.

The structuredness considers the coupling between the software parts and the affects to software analyzability by identifying the scope of the error inspection process.

The accountability tracks every action occurred in the system [16], which allows to identify the errors and it is causes efficiently. Moreover, it decrees the relationships between the components reduced the scope affected by error, which reduce the area of the inspection.

The instrumentations simplify the task and reduce the time and the efforts. Moreover, the instrumentation increases the accuracy of the software analyzing.

The legibility of the system simplifies the job of identifying the required modification.

C. *Modifiability*

One of the greatest challenges facing software engineers is the management of change control. It has been estimated that the cost of change control can be between 40% and 70% of the life cycle costs [17]. Software maintenance is a modification of software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changing environment.

Changeability is the ability of the software product to change, by allowing developers to modify the delivered system and characterize the effects using different criteria through its limited available information.

In order to simplify the process of software modification, and to minimize the effect of the modification on the other parts of the system, several characteristics are required. Figure 3 shows the characteristics of the system that are required to present the system modifiability.

Software stability describes the ability of the software to avoid or minimize unexpected effects from system modifications. This characteristic is considered in terms of coupling and coherence in software modularity.

In order to increase the expandability and reduce the effort to modify the system, the generality is considered. The generality is a level of abstraction to retrieve results based on desired generality appropriate for a user's knowledge and interests.

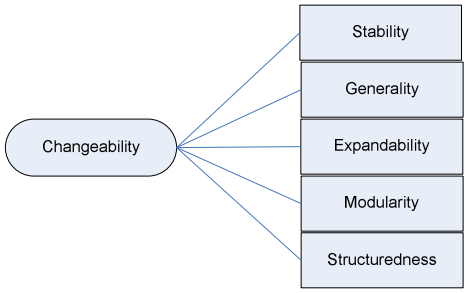

Figure 3: Changeability Factors

The expandability represents the ability of the system to grow. In order to add new function to the existing system, the expandability is very important.

The modular software allows each part of the system working independently with minimum effects on the other parts of the system.

Software structuredness is very important in terms of modification scope, which considers the coupling of the software components.

D. *Testability*

Software testability is the ability to test whether the software meets its requirements. In maintenance process, the testability represents whether the modification has achieved its goals and if there is any side effects of the this modification. Testability reduces the defects resulted from poorly software design. The test process achieves by providing some type of individual and group evaluation, in order to check the functionality of the system components and the integrated system to insure it performs required functions. Figure 4 shows the software characteristics that help in this process.

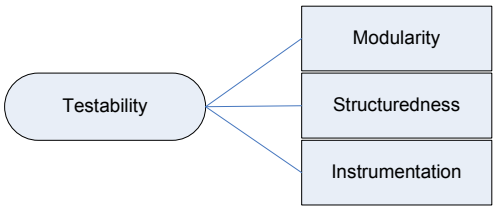

Figure 4: Testability Factors

The modularity allows to evaluates each part of the system independently and then evaluate the integration between these parts. This way of testing simplifies and increases the errors inspections and the reasons causing the errors.

The reducing of the relationships between the components within the system considered in structuredness factor is important; in order to reduce the efforts and the complexity required testing the components and their relations.

The instrumentations used on the testing increase the accuracy of errors inspections and decrease the efforts and the time required to test the system.

V. THE MAINTENANCE FRAMEWORK

As discussed earlier, the maintenance stage consists of four main processes, software understanding, analyzing, modifying, and testing. Figure 5 shows the processes of the maintenance

and the required quality characteristics for each process. Software maintenance process consists of four main sequential processes, whereas the developer able to back to the modification process if there is any error inspected in the test process. Furthermore, one more factor, software compliance is added. This factor presents whether those processes are covered by any international standard or certificates. The compliance can be considered for the whole processes or even for each one of them independently. Figure 6 shows the structure of the software maintainability including the sub-characteristics and its relationships.

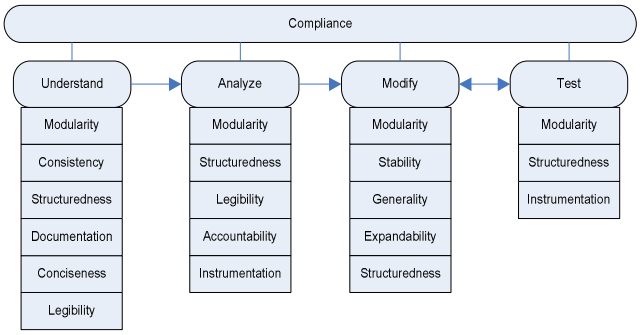

Figure 5: The Processes of Software Maintenance

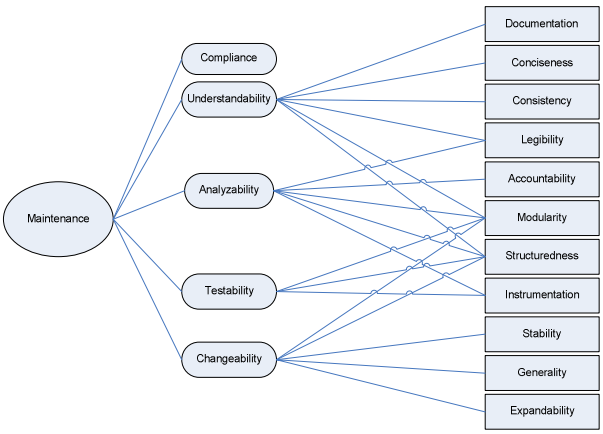

Figure 6: Software Maintainability Characteristics

VI. Conclusion

From different perspectives, the maintainability of the software is discussed, and different software characteristics are considered to calculate the maintainability factor. This paper discusses the processes of software maintenance and the concept of the software maintainability based on the maintenance processes. Eleven software product characteristics that affect the four tasks of software maintenance were found. Furthermore, in order to evaluate whether the maintenance process or some of its task follows any standard or international certificate, the compliance factor is considered.

References

[1] S. C. B. D. Souza, *et al.*, "A study of the documentation essential to software maintenance," presented at the Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information, Coventry, United Kingdom, 2005.

[2] P. Bhatt, *et al.*, "Dynamics of software maintenance," *SIGSOFT Softw. Eng. Notes,* vol. 29, pp. 1-5, 2004.

[3] S. Das, *et al.*, "Understanding documentation value in software maintenance," presented at the Proceedings of the 2007 symposium on Computer human interaction for the management of information technology, Cambridge, Massachusetts, 2007.

[4] E. Burch and K. Hsiang-Jui, "Modeling software maintenance requests: a case study," in *Software Maintenance, 1997. Proceedings., International Conference on*, 1997, pp. 40-47.

[5] P. F. Tiako, "Maintenance in joint software development," in *Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International*, 2002, pp. 1077-1080.

[6] C.-C. Chiang and C. W. Ford, "Maintainability and reusability issues in CORBA-based systems," presented at the Proceedings of the 43rd annual Southeast regional conference - Volume 2, Kennesaw, Georgia, 2005.

[7] W. Hordijk and R. Wieringa, "Surveying the factors that influence maintainability: research design," presented at the Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering, Lisbon, Portugal, 2005.

[8] H. Mittal and P. Bhatia, "Software maintainability assessment based on fuzzy logic technique," *SIGSOFT Softw. Eng. Notes,* vol. 34, pp. 1-5, 2009.

[9] Y. Singh, *et al.*, "Predicting software maintenance using fuzzy model," *SIGSOFT Softw. Eng. Notes,* vol. 34, pp. 1-6, 2009.

[10] K. K. Aggarwal, *et al.*, "An integrated measure of software maintainability," in *Reliability and Maintainability Symposium, 2002. Proceedings. Annual*, 2002, pp. 235-241.

[11] V. Basili, *et al.*, "Understanding and predicting the process of software maintenance releases," in *Software Engineering, 1996., Proceedings of the 18th International Conference on*, 1996, pp. 464-474.

[12] K. Laitinen, "Estimating understandability of software documents," *SIGSOFT Softw. Eng. Notes,* vol. 21, pp. 81-92, 1996.

[13] S. Huang and S. Tilley, "Towards a documentation maturity model," presented at the Proceedings of the 21st annual international conference on Documentation, San Francisco, CA, USA, 2003.

[14] A. Kumar, *et al.*, "A quantitative evaluation of aspect-oriented software quality model (AOSQUAMO)," *SIGSOFT Softw. Eng. Notes,* vol. 34, pp. 1-9, 2009.

[15] P. S. Grover, *et al.*, "Few useful considerations for maintaining software components and component-based systems," *SIGSOFT Softw. Eng. Notes,* vol. 32, pp. 1-5, 2007.

[16] E. Bertino, *et al.*, "End-to-end accountability in grid computing systems for coalition information sharing," presented at the Proceedings of the 4th annual workshop on Cyber security and information intelligence research: developing strategies to meet the cyber security and information intelligence challenges ahead, Oak Ridge, Tennessee, 2008.

[17] B. Hunt, *et al.*, "Software Maintenance Implications on Cost and Schedule," in *Aerospace Conference, 2008 IEEE*, 2008, pp. 1-6.