Investigating Software Maintainability Development: A case for ISO 9126

Ahmad Haboush, Mohammad Alnabhan, Anas AL-Badareen, Mohammad Al-nawayseh and Bassam EL-Zaghmouri

Faculty of Information Technology, Jerash University
Jerash, Jordan

Abstract

Software maintainability has been considered as a main characteristic in many software product quality models. However, these models have different definitions for maintainability and sub characteristics. ISO 9126 is one of the main and most frequently used models in software product quality. This model has been revised and replaced by ISO 25010 as a new model of software product quality. In addition to the many modifications that were performed on ISO 9126 model, maintainability was one of the main modified characteristics. However, it was developed unclearly without any standard base, and with no clear definition or evidence of how the sub characteristics were defined and modified. This paper investigates these modifications and the differences between the definitions of the maintainability in the two models, ISO 9126 and ISO 25010. As a result of this discussion, it has been concluded that both models ISO 9126 and ISO 25010 lack of a clear definition or standard base for defining software maintainability and its sub characteristics.

Keywords: *Quality Model, ISO/IEC 9126, ISO/IEC 25010, Maintainability, Reusability.*

1. Introduction

The In last decades, many models of software quality have been proposed. In 1978, ISO/IEC intended to propose a standard model for software quality in order to unify the evaluation process of software quality as well as to eliminate the debate between software quality models. The first version was released in 1991, and called ISO 9126.

As shown in figure 1, the model specified six characteristics including Functionality, Reliability, Usability, Efficiency, Maintainability, and Portability; which are further divided into 21 sub-characteristics. The defined characteristics are applicable to every kind of software, including computer programs and data contained in firmware, and provide consistent terminology for software product quality. They also provide a framework for making trade-offs between software product capabilities. Several corrections and enhancements were performed and revised versions were released, as follows:

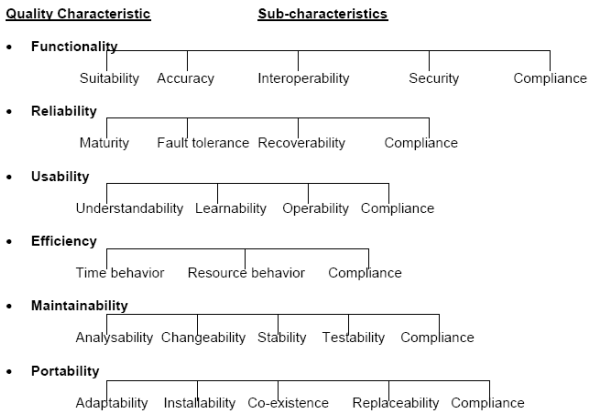

Fig. 1: The ISO/IEC 9126 Model

- ISO/IEC 9126-1 [1]: new updated quality model.
- ISO/IEC 9126-2 [2]: new set of external measures.
- ISO/IEC 9126-3 [3]: new set of internal measures.
- ISO/IEC 9126-4 [4]: new set of quality in use measures.

However, this model suffers from the ambiguity in the definition of the quality characteristics and their relationships [5], and it is not suitable to measure the design quality of software product [6]. The inconsistency in definitions of the quality characteristics and their sub-characteristics results contradictions in the developed models [7]. Consequently, different quality models are developed to evaluate same types of software products. Therefore, ISO/IEC proposed a new model that aim to resolve these problems and many others as stated in literature.

ISO/IEC 25010 [8] is a derived version from ISO/IEC 9126:1991 (see figure 2). The model incorporates the same quality characteristics with some modifications. Software maintainability is one of the main characteristics that were

affected by these modifications. Two characteristics were eliminated (stability and compliance) and new other two were added (reusability, and modularity).

However, as in ISO 9126, the maintainability sub characteristics were defined ambiguously without clear definition or standard form, the modifications in the new version ISO 25010 were also performed in the same way.

The paper starts in section two with presenting the maintenance process and its objectives. Section three presents the reuse process and its relationship with the maintenance. Section four presents the modularity characteristic and its relationship with maintenance processes. Section five presents the stability and its relationship with modifiability. Section six presents the compliance characteristic. Finally, discussion and recommendations are presented in section seven.

2. Software Maintenance

Software maintenance is the process of modifying software product either for correction, enhancement, or adaptation. It is an inescapable part of software lifecycle [9], which is required to keep the software product useful and updated with the world changes [10-11]. However, the main problem with software maintenance is that it is the most hard, costly and error and error-prone process in software life cycle [12-13]. Its cost is approximately equal to (80-90%) of the total cost of software development life cycle [10, 14]. Therefore, much attention has been given to this process, and how it can be performed efficiently with a minimum cost.

The maintenance process has been defined as one of the main factors that have impact on the quality of the software product. The ability to achieve this process has been considered early since the first model of software quality. All the models agreed that, software maintainability is one of the main factors in software quality, but they differ in the characteristics that are used to measure this factor and the structure of these characteristics. Generally, software maintenance requires an understandable, analyzable, modifiable, and testable software product in order to be performed efficiently.

According to ISO/IEC 25010 [8], software maintainability is the degree of the effectiveness and efficiency of modifying the software product by maintainers. This characteristic represents the ability to modify the software product efficiently and easily in order to correct, enhance, or adapt it. ISO 9126 involves five sub characteristics in order to measure the maintainability factor: analyzability, changeability, stability, testability, and compliance, while ISO 25010 involves modularity, reusability, analyzability, modifiability, and testability.

According to IEEE [15] software maintainability is any modification made on software product after delivery, in order to correct faults, improve performance and other attributes, or to adapt the product to a modified environment. This standard identified four main objectives that the maintenance process can be performed for:

a) Corrective;
b) Adaptive;
c) Perfective; and
d) Emergency.

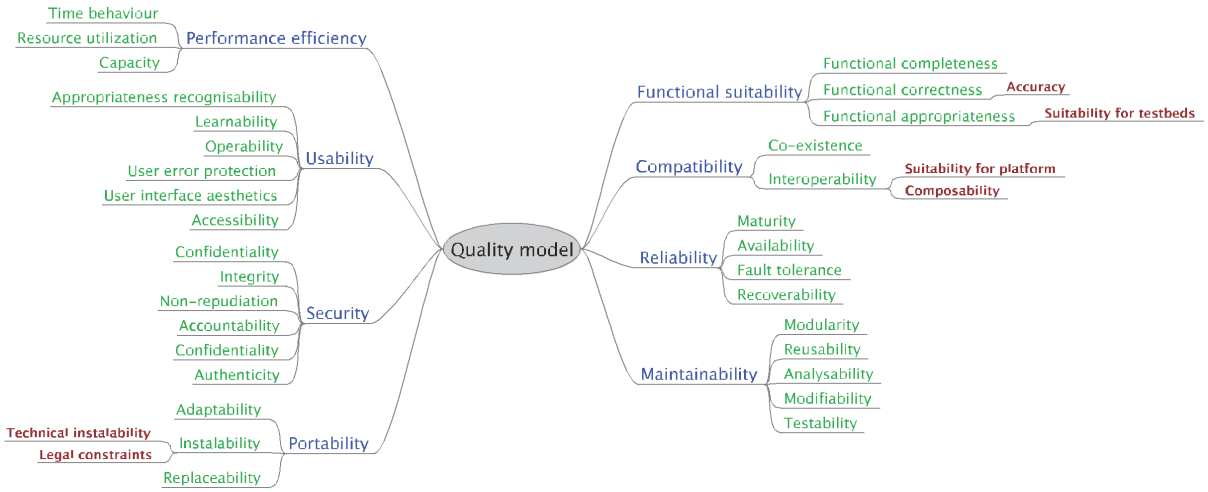

Fig. 2: The ISO/IEC 25010 Quality Model

According to AL-Badareen [16-17], software maintenance is a process of modifying software product, which conducted through four main tasks: understanding, analyzing, modifying, and testing the software product, see figure 3.

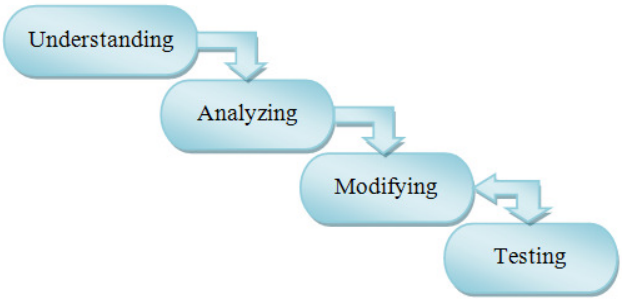

Fig. 3: Software Maintenance Processes

Software maintenance mainly focuses on the modification of software product. According to Souza [10], the maintenance process is performed on software product after its delivery. The maintenance process does not care about modifying the software product, as much as achieving the modification objectives efficiently.

3. Software Reuse

Software reuse is the process of using an existing software product or part of it in order to develop a new software product. This process is used to reduce the effort, cost, and time of developing software products [18]. It increases the productivity of software development [19] [20] as well as enhances the quality it[21].

According to ISO/IEC 25010 [8], software reusability is the ability of the software asset to be used in developing more than one software product or other assets. Reusability also is the degree to which a thing can be reused [22]. It represents the ability to use a part or the whole system in other systems [23-26] which are related to the packaging and scoping of the functions that programs perform [27].

For any software asset, two main conditions must be achieved in order to be used in the new system: the ability to achieve the intended functions in the new system, and the ability to be adapted to the new architecture and work with the components in the new system. Therefore, to define any software asset as reusable, it should be able to work with different types of software components in different systems and environments. These abilities can be achieved by considering them early in developing the reusable assets or modifying the existing components to achieve them. Therefore, three main types of reusable assets can be involved in the reuse process [18]: normal asset, reusable asset with internal components, and reusable asset with market components.

- **Normal asset:** is a software component developed for specific function in certain software product meanwhile the reusability characteristics are not considered during the development process. This type of software components is in-house developed software which includes design and its all internal components. Moreover, it is required to be portable and interoperable to work with different systems in different environments. Therefore, it has to be modified to achieve the new requirements.

- **Reusable Asset with Internal Components:** software components that are developed to be used in different software systems in the future. It is in-house developed software which includes the design and all of its internal components. This type of components possesses the reusability characteristics, and also allows any modification might be required for adaptation to the new system.

- **Reusable Asset with Market Components:** is reusable software components imported from external sources. It is a market resource, which does not allow any modification might be required. Therefore, any modification required for adaptation has to be on the system architecture instead of the component.

Based on the types of the reusable assets, the reuse process can be performed in two main ways: white box, and black box. White box reuse is the process of modifying existing software assets in order to fit the new requirements in the new system. This process is performed based on the in-house developed software assets, for both: normal asset to achieve the new requirements or the reusable assets with the internal components that might require a modification for adaptation. Black box reuse is the process of using existing software products without any modification. The process can be performed on any type of software asset that might not need or allow any modification. This process is compulsory for the reusable assets with market components, which it does not allow any modification.

Consequently, software reuse care about the ability of the software assets to achieve the requirements of the new

systems and to be used efficiently. The main objective of the reuse process is to use the software asset in the new system, and it does not matter whether it is able to be modified as much as to achieve the requirements in the new systems. The modification is required only in the white box reuse, whereas the black box reuse does not need or allow any modification. Therefore, the relationship between reusability and maintainability depends on the type of software assets and the reuse process, which can be identified as follow:

Is the reusable software maintainable?

In the white box reuse, software asset has to be maintainable, which is required to be modified for the new system's and environment's requirements. But, in the black box reuse, especially the reusable assets with market components, software asset is not allowed to be modified, and therefore, it should not be maintainable.

Is the maintainable software reusable?

The main idea of the white box reuse is to modify existing software asset in order to achieve the new system's requirements. Therefore, any software asset has the ability to be modified to achieve the new system requirements can be reusable.

4. Modularity

Modularity is an available method to solve a complex problem, which aims to decompose and integrate all objects [16]. It is the degree of using independent software components, so any change in one component has a minimal impact on other components in the system. This method concerns about decomposing the system into manageable components, which allows:

- Understanding each part of the system independently, instead of understanding the whole system as one part.
- Analyzing a system efficiently by identify which part of the system requires a modification, instead of analyzing the whole system at the same time.
- Modifying the system's components without affecting other components and identifying the parts that are affected from the modification efficiently.
- Testing the modified parts instead of testing the whole system.

Modularity is an important characteristic required in every task of software maintenance. Therefore, ISO 25010 considered it as sub factor along with analyzability, modifiability, and testability. However, the total value for maintainability is not affected in this modification. But the problem in the maintainability sub characteristics: analyzability, modifiability, and testability, that they miss one of their main sub characteristics, which is modularity. IEEE Standard for Software Quality Metrics indicated that the quality sub-characteristics are more meaningful to the technical personnel , it also facilitates objective communication between them and their managers [28]. Therefore, the quality sub characteristics, analyzability, modifiability, and testability are meaningless for the technical personnel, although the modularity value is presented independently. Consequently, a miscommunication will occur between technical personnel and managers.

5. Stability

Modifiability is the degree of modifying software products efficiently and effectively without any side effects or affecting the quality of the system. Software modification affects the behavior of the software components, which might cause new problems by achieving the intended objectives. Therefore, software system has to be able to avoid or reduce any unexpected effects that might arise from modifications.

However, in order to perform the modification efficiently it has to be stable. This characteristic was defined as sub factor of software maintainability along with modifiability in ISO 9126, which is different from the definition of the modifiability. Moreover, this indicates that the modifiability concerns about the ability of software product to be modified, and it does not matter whether it affects the other parts of the system or not. ISO 25010 mentioned that the modifiability is a combination of changeability and stability, and therefore, this characteristic is removed from the sub characteristics level of the maintainability factor.

6. Compliance

According to ISO 25010, the compliance sub-characteristics have been removed. as compliance with laws and regulations is part of the overall system requirements, rather than specifically part of software quality [8]. Following any standard is to conform that certain level of quality has been achieved.

7. Discussion

This paper discussed changes in software maintainability from ISO 9126 to ISO 25010. It includes four sub

characteristics, two eliminated (stability and compliance) and two added (reusability and modularity). The study was intended to investigate the validity of the relationships between those sub characteristics and the maintainability factor. The discussion concluded the following:

Maintainable software can be reusable but the reusable software might not be maintainable. That is, in black box reuse, software assets are not required to be modified. The reusable assets have to be portable and compatible, whereas, software maintenance intends to make the software assets portable and compatible. Therefore, the enhancement of software portability and compatibility are added values to the reusability, whereas, the portable and compatible software not should be maintainable.

However, software maintainability represents whether the software product is able to be modified in order to correct, enhance, or adapt the software. Software reusability represents whether the software is able to be used in other software products, which it does not matter whether it requires a modification or not. Two types of reusable software are required to be maintainable in order to fit the new systems' requirements, regarding the functionality, compatibility, and portability issues.

Software modularity is an important characteristic that significantly affects the maintenance process. It is required to understand, analyze, modify, and test the software product. Including the modularity as a sub factor along with these characteristic do not affect the total value of the maintainability, but it affects the values of those sub characteristics. These sub characteristics will be meaningless, by missing one of their main sub characteristics.
The importance of stability is not less than the other characteristics of software modifiability. Where the side effects caused by software modification might make this process useless and harmful instead of being useful.

Compliance is only considered as an indicator that the characteristic of software product were achieved in a certain level of standard, and it should not completely satisfy the quality standard.

8. Conclusion and Recommendation

This study discussed the development of software maintainability characteristic in ISO 9126 and ISO 25010. The study presented the definition of software maintainability and its sub characteristics in both standards. Moreover, discussed the differences between the sub-characteristics were used to evaluate the maintainability in both standards, how and why these sub characteristics were included and excluded from the standards. The results of the discussion show that the sub characteristics of the maintainability characteristic were included and excluded in both versions of ISO subjectively. There is no clear definition or justification of include and exclude these sub characteristics.

Consequently, maintainability characteristic was developed unclearly without any standard base, in both ISO 9126 and ISO 25010. The ambiguity of developing quality characteristics and their relationships make the models confusable, questionable, and consequently debatable. Therefore, there is a crucial need for a standard base for developing and decomposing software quality characteristics, instead of developing new quality models. At this time only, a new model of software quality can be developed and achieve its intended objectives to resolve the debate among software quality models, which is the main idea of ISO/IEC JTC1 since 1978.

References

[1] ISO/IEC, "IEC 9126-1: Software Engineering-Product Quality-Part 1: Quality Model," in *Geneva, Switzerland: International Organization for Standardization*, ed, 2001.
[2] ISO/IEC, "ISO/IEC TR 9126-2: Software engineering-software product quality-part 2: External metrics," in *Geneva, Switzerland: International Organization for Standardization*, ed, 2003.
[3] ISO/IEC, "ISO/IEC TR 9126-3 Software engineering-software product quality-part 3: Internal metrics," in *Geneva, Switzerland: International Organization for Standardization*, ed, 2003.
[4] ISO/IEC, "ISO/IEC DTR 9126-2 Software engineering – software product quality-part 4: Quality in use metrics," in *Geneva, Switzerland: International Organization for Standardization*, ed, 2001.
[5] B. Kitchenham and S. L. Pfleeger, "Software quality: the elusive target [special issues section]," *Software, IEEE,* vol. 13, pp. 12-21, 1996.
[6] H. Al-Kilidar, *et al.*, "The use and usefulness of the ISO/IEC 9126 quality standard," 2005, p. 7 pp.
[7] A. B. Al-Badareen, *et al.*, "Software Quality Models: A Comparative Study," in *Software Engineering and Computer Systems*, ed: Springer, 2011, pp. 46-55.
[8] ISO/IEC, "Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models " in *ISO/IEC 25010*, ed. IEC, 2011.
[9] M. Kernahan, *et al.*, "Extracting Traceability Information from C# Projects," in *WSEAS International Conference on ENGINEERING EDUCATION*, Athens, Greece, 2005.

[10] 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.

[11] U. Vora and N. Sarda, "Framework for evolving systems," in *5th WSEAS International Conference on SIGNAL PROCESSING, ROBOTICS and AUTOMATION (ISPRA '06)*, Madrid, Spain,, 2006, pp. 145-150.

[12] 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.

[13] A. H. Mohamed, "Facilitating Tacit-Knowledge Acquisition within Requirements Engineering," in *10th WSEAS International Conference on APPLIED COMPUTER SCIENCE (ACS '10)*, Iwate Prefectural University, Japan, 2010, pp. 27-32.

[14] 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.

[15] IEEE, "IEEE Standard for Software Maintenance," in *IEEE Std 1219-1998*, ed: IEEE, 1998.

[16] A. B. AL-Badareen, *et al.*, "The Impact of Software Quality on Maintenance Process," *International Journal of Computers,* vol. 5, pp. 183-190, 2011.

[17] A. B. AL-Badareen, *et al.*, "Software Quality Evaluation through Maintenance Processes," in *NAUN Conference of CONTROL*, Puerto De La Cruz, Tenerife, Spain, 2010.

[18] A. B. AL-Badareen, *et al.*, "Reusable Software Components Life Cycle," *International Journal of Computers,* vol. 5, pp. 191-199, 2011.

[19] P. Mohagheghi and R. Conradi, "An empirical investigation of software reuse benefits in a large telecom product," *ACM Trans. Softw. Eng. Methodol.,* vol. 17, pp. 1-31, 2008.

[20] I. PHILIPPOW, "Utilization of Object-Oriented Models," in *WSES International Conference on Multimedia, Internet, Video Technologies 2001*, Malta, 2001.

[21] A. Sharma, *et al.*, "Reusability assessment for software components," *SIGSOFT Softw. Eng. Notes,* vol. 34, pp. 1-6, 2009.

[22] W. Frakes and C. Terry, "Software reuse: metrics and models," *ACM Comput. Surv.,* vol. 28, pp. 415-435, 1996.

[23] J. A. McCall, *et al.*, "Factors in Software Quality," *Griffiths Air Force Base, N.Y. Rome Air Development Center Air Force Systems Command,* 1977.

[24] N. S. Gill, "Reusability issues in component-based development," *SIGSOFT Softw. Eng. Notes,* vol. 28, pp. 4-4, 2003.

[25] C. Luer, "Assessing Module Reusability," in *Assessment of Contemporary Modularization Techniques, 2007. ICSE Workshops ACoM '07. First International Workshop on*, 2007, pp. 7-7.

[26] F. Haiguang, "Modeling and Analysis for Educational Software Quality Hierarchy Triangle," in *Web-based Learning, 2008. ICWL 2008. Seventh International Conference on*, 2008, pp. 14-18.

[27] J. J. E. Gaffney, "Metrics in software quality assurance," presented at the Proceedings of the ACM '81 conference, 1981.

[28] IEEE, "IEEE Standard for a Software Quality Metrics Methodology," in *IEEE Std 1061-1998 (R2009)*, ed, 1998, p. i.