International Journal of Engineering Research and Technology. ISSN 0974-3154, Volume 13, Number 1 (2020), pp. 158-162
© International Research Publication House. https://dx.doi.org/10.37624/IJERT/13.1.2020.158-162

Software Quality Model for Maintenance Software Purposes

Hamed Fawareh

Department of Software Engineering, Zarqa University, Zarqa, Jordan.

ORCID: 0000-0002-0853-3149

Abstract: Software maintenance tools developed for attempted to raise the success rate of software systems over the past century. Improve software tools quality models and other software elements to make it more customer satisfaction and achieve customer permanence. Several quality models and variables have proposed to decrease software system failure and complexity. Also, software quality models proposed to assess the general and particular types of software products. These models have proposed to determine the general or particular scopes of software products - none of these quality models concerns the quality of software maintenance tools. The proposed software quality maintenance models developed based on the maintenance tools factor and the comparisons between the well-known quality models. These comparisons are the leakage of criteria based on distinct views and knowledge of maintenance tools requirement. The proposed technique applied to software maintenance tools. The outcome of the proposed technique demonstrates that the twelve factors must deem to increase the quality of software maintenance tools.

Keywords: software quality, quality factor, software maintenance, software maintenance tool.

INTRODUCTION

The progress of software quality models during the past years concerns the development process only; there are still problems that stop them from being commonly adopted in maintenance and reengineering tools in practice. Maintainers in practice, disappointed because quality models do not satisfy maintenance expectations. It often uncertain continue how quality models in practice can evaluate and predict quality tools used during reengineering tools. The last three decades of quality modelling produce a variety of quality strategies and factors.

Software quality plays a vital role in the overall software system's success; it considered an essential aspect for maintainer, users and managers of projects. Success is found relatively rare in the world of software projects. One potential reason might be the difference in the original software and maintained software of the meaning of success in the minds of people evaluating the quality of the project. Therefore, the criteria for maintenance project success, as believed by various stakeholder groups, do not match. The highest determining factor of achievement is the functionality and quality of the project outcome, success in external goals such as customer satisfaction. Maintenance factors are essential to the acceptance of software maintenance tools and become commonly used.

Completion software is often far from meeting user expectations and business performance objectives. The software project success or failure is internal process measure of the project team's performance, including criteria such as scheduling, budgeting, meeting the project's technical objectives and maintaining smooth working relationships within the team and parent organisation.

Based on a literature review and interviews with seasoned project maintainer, three distinct aspects of project maintenance and efficiency established as the metrics against which to assess a project's success or failure. These aspects are:

- The process implementation.
- The value of perceived project.
- The satisfaction client for project delivered.
- The maintenance tools.

The first of these aspects is primarily concerned with the internal efficiency of the project implementation process, which reflect on the maintenance phase. The second aspect of project success or failure assessment is the perceived project quality; it includes the perception by the project team of the value and usefulness of the outcomes of the project. This evaluation emphasises the potential impact of the project on users — the judgment of the project team as to how good a job they have done for the client. The evaluation and maintenance of the project by the project team may or may not agree with the evaluation and maintenance of the client. The third aspect of project performance, customer satisfaction, is an external measure of customer effectiveness [1].

SOFTWARE QUALITY MODELS

Over the past years, the production of high-quality software has considered an important topic that has discussed. Software quality factor identified as the criteria that will cover all software characteristics and software usage elements to ensure complete user satisfaction [2-3].

Quality models are used in conjunction with software factor to identify a high-quality software product in addition to evaluating the feature of the software output [4].

In the literature of software engineering several quality models have been proposed and gradually evolved such as McCall quality model, Boehm quality model, Dromey quality model, FURPS quality model and ISO 9000 quality model, each model contains different quality characteristics or factors [5-6]. These models have suggested maintenance as a specific type of software products, while maintenance is a spate phase [7]. The following subsection discusses the models in details.

McCall Model: proposed the first model in 1977, which defines the qualities of the software product as a hierarchy divided into three main components: factors, criteria and metrics. The factors represent the system feature, a quality criterion is an attribute of software production and design-related quality factor, and metrics defining and using a measurement scale and method [8].

McCall model contains eleven factors and twenty-three criteria; these factors divided into three groups of products, namely: Transition, Revision and Operations. According to Lee [8], McCall mentions maintainability as a factor for software development. Because this model is ancient, there was no consideration for new features of maintenance tools; it has not taken into account the unique characteristics of software maintenance tools [9],[5].

Boehm Model defined the primary quality characteristic as a general utility. The main major of Boehm model is to address the weaknesses of models that evaluate software quality automatically, and this model gives quantitative results indicating software quality. Boehm model discussed the high-level characteristics and classified it into three groups, namely: Utility, Maintenance and Portability [7].

Seven qualities collectively characteristics exemplify the qualities predictable from a software system portability, reliability, efficiency, usability, testability, comprehensibility, flexibility and human engineering [10-11].

Alternative Models: two alternative models of the classic McCall model proposed which are: Evans and Marciniak model, this model emerged as alternatives to McCall's classic model proposed by Evans and Marciniak in 1987. The new model eliminated the testability factor from McCall model and added two new factors which are Verifiability and Expandability. Thus, the model became composed of twelve quality factors, partitioned to three groups, namely adaptation, performance and design.

Deutsch and Willis model: another alternative of McCall's classic model proposed in 1988. Also, this model as Evans and Marciniak model excluded testability factor from McCall classic model and added Safety, Manageability and Survivability factor is to the new model. Deutsch and Willis's quality model comprises of fifteen factors divided into four functional classifications, performance, change and management. Youness show a comparison between the classic McCall model and alternative model, [6]. Both alternative models consider maintenance as one factor of software quality, while maintenance should consider as a spate phase.

Dromey Model proposed a framework in 1995 for assessing the requirements, designing, and implementation of the system. He noted that the assessment is various for each software product, so we need a dynamic modelling idea. The main objective of this model was to obtain a model that fits all types of software systems and realise the relationship between quality characteristics and sub-characteristics [12]. Several attributes define Dromey models, such as two layers of attributes, high-level attributes, and subordinate attributes. One of the drawbacks of this model is a shortage of software quality measurement criteria [13].

FURPS Model proposed a quality model called FURPS, which are mean F: Functionality, U: Usability, R: Reliability, P: Performance and S: Supportability. In this model, the features are taxonomy into two groups: The inputs and expected outputs defined as the functional requirements of the software system. The desired attributes of the software system are known as non-functional requirements, such as reliability and usability. The model fails to take into account some features of a software product such as portability [5].

ISO 9000 Model: The International Organization for Standardization (ISO) proposed ISO 9000 model in 1991, which considered the foundation of quality assurance. The quality characteristics of the software product are the structures hierarchically classified as characteristics and sub characteristics [14]. Quality factors at the top of the hierarchical and the criteria of a software product are at the lowest level. ISO 9000 model consists of six factors which divided into twenty-seven sub-characteristics. The defined factors in this model can apply to all software types, including firmware computer programs and data, and it can provide consistent software product quality terminology. It also provides a structure that helps stakeholder to trade-offs between software product capacities [15, 5]. Lately, the ISO/IEC 205010:2011system and software product quality model replaced ISO/IEC 9126-1: 2001 software product quality model, which include eight quality factors: "Functional suitability, reliability, operability, security, performance efficiency, compatibility, maintainability, and portability. The twenty-eight quality factors arrange in eight quality characteristics" [5].

MAINTENANCE FACTOR

In the software engineering field, software quality is one of the oldest approaches used by the software researches. Quality models are the approach for comprehension and manipulating an issue in engineering and science disciplines [2]. Quality models have, therefore become a well-accepted means of describing and managing software quality. Starting with the hierarchical models suggested by Boehm et al. [11], Different quality models have developed over the last 20 years, some of which have standardised. Some of these models used in different aspect of software life cycle purposes, For example, to help specify the quality requirements, to evaluate existing systems or to predict the fault density of a system in the field [3].

This paper proposed a software quality model for maintenance quality factors. These factors are essential for the quality software system. Also, It plays the main rules in the success and failure of the software systems. In this study, we specify the main factor affecting the success and failure of software maintenance by analysing the definition of software maintenance. According to Priyadrshi and Kshirasgar [16] maintenance activities are divided into four groups, Corrective maintenance to correct failures: processing failures and performance failures, adaptive maintenance to enable the system to adapt to changes in its data environment or processing environment, perfective maintenance to make a variety of improvements, namely, user experience, processing efficiency, and maintainability, and preventive maintenance to prevent problems from occurring by modifying software products.

Chapin et al. [17] divide the maintenance activities into twelve types of maintenance activities, and factors were grouped into four clusters, as shown in table 1. We will consider the twelve factors to satisfy the maintenance quality for software maintenance.

Table 1: Maintenance Factors

Cluster	Type
Business Rules	Enhancive
	Corrective
	Reductive
Software Properties	Adaptive
	Performance
	Preventive
	Groomative
Documentation	Adaptive
	Reformative
Support Interface	Evaluative
	Consultive
	Training

Based on a comparison of all previous elements and a repeated cancellation or that gives the same meaning, we have got a set of maintenance factors that appear in table 2 base on the classification of software maintenance and evolution developed by Chapin et al. [17]. According to analysis results, we classified a maintenance factor from software engineering quality into two categories

• Included in the previous models

• None Included in the previous models

In this section, the factors that include in previous software quality model exclude, and the factors that are related to software engineering quality were focused on, as shown in the following table 2.

Table 2: Factor Definitions

Factor	Definition	Related Factor
Training	This means training the stakeholders about the implementation of the system.	--
Consultive	In this type, cost and length of time are estimated for maintenance work, personnel run a help desk, customers are assisted to prepare maintenance work requests, and personnel make expert knowledge about the available resources and the system to others in the organization to improve efficiency.	--
Evaluative	In this type, common activities include reviewing the program code and documentations, examining the ripple effect of a proposed change, designing and executing tests, examining the programming support provided by the operating system, and finding the required data and debugging.	--
Reformative	Ordinary activities in this type improve the readability of the documentation, make the documentation consistent with other changes in the system, prepare training materials, and add entries to a data dictionary.	--
Updative	Ordinary activities in this type are substituting out-of-date documentation with up-to-date documentation, making semi-formal, say, in UML to document current program code, and updating the documentation with test plans.	--
Groomative	Ordinary activities in this type are substituting components and algorithms with more efficient and simpler ones, modifying the conventions for naming data, changing access authorizations, compiling source code, and doing backups.	Modifiability
Preventive	Ordinary activities in this type perform changes to enhance maintainability, and establish a base for making a future transition to an emerging technology.	Maintainability
Performance	Activities in performance type produce results that impact the user. Those activities improve system up time and replace components and algorithms with faster ones.	Performance
Adaptive	Ordinary activities in this type port the software to a different execution platform, and increase the utilization of COTS components.	--
Reductive	Ordinary activities in this type drop some data generated for the customer, decreasing the amount of data input to the system, and decreasing the amount of data produced by the system.	--
Corrective	Ordinary activities in this type are correcting identified bugs, adding defensive programming strategies, and modifying the ways exceptions are handled.	correctness
Enhancive	Ordinary activities in this type are adding and modifying business rules to enhance the system's functionality available to the customer, and adding new data flows into or out of the software.	--

QUALITY CRITERIA

Quality criteria and their relationship to quality factors represent the central part of evaluating and defining any quality factor. These criteria may be attributes of the product or attributes of the production process [21].

Studying and analysing the relationship between maintenance definition and software quality, we proposed software quality factors related to maintenance elements and its criterion. Most organisations are concerned with the quality of the software systems used within their organisations. Therefore, the measurement and evaluation of the quality of software systems are essential. Table 2 shows the relationship between criteria definitions and related software quality factors.

In this section, we focus on the software business rules, which is particularly interested in measuring customer satisfaction with the quality software system from the maintenance aspect. The maintenance factor defines as a set of criteria (business rules, software properties, documentation and support interface). Table 1 classified the criteria into four main groups based on their definition.

To measure the criteria of the maintenance factor and provide quantitative values to the stakeholder, which enable him to assess the quality of the product in the maintenance aspect in the decision-making process. We will study the business roles as a case study for measuring the new factors.

In this study, we used function point as a method for measuring the size and productivity of software systems. It also used to calculate the size and complexity of applications based on outputs, inputs, queries, internal files and interfaces.

To calculation the business rules complexity of the software system for satisfactions of software failure and permanence variables, we set up the following definition for a business rules metric:

- Basic Activities in a system (NOBA): NOBA metric counts necessary activities in a system. NOBA is a simple one-dimensional metric based on a function point activities, unlike other complexity metrics which manipulate two or more dimensions of a process.
- Basic Structured Activities (NBSA): NBSA calculates how deeply we used (length). NOBA metric is another simple one-dimensional length metric similar to NOBA. However, instead of counting necessary activities, it counts the number of structured activities in a system. It should note that NBSA counts the number of structured activities and attach weights.

Information Flow complexity (IFC): IFC metric is an adaptation on a system. It is a fan-in represented by input activities while output activities represent fan-out. The IFC defined as the square of the product of the Number of Input Activities (NOIA) and the Number of Output Activities (NOOA) contained in it.

The product metrics is particularly interested in measuring the criteria of maintenance tools factor. We determined a set of factor criteria related to software maintenance. As we mentioned earlier, we divided the maintenance factor criteria into four groups according to their similarity in the measurement method. We measure each of maintenance factor criteria, to provide quantitative values, which help stakeholders to assess the quality of the software maintenance system to make the decision.

To measure and evaluate the software system regarding supporting the maintenance factor, we proposed a set of rules that must abide by to help evaluate the quality of the software system regarding the maintenance as follows:

We used function point as a method for measuring the size and productivity of software systems based on outputs, inputs, queries, internal files and interfaces.

Maintenace Information Flow (IF): IF metric is an adaptation of business rules in a system. The IF is defined as; square the result of the Number of Input (NI) by the Number of Output (NO) in the system. These can show in Eq. Number 1 and Eq.Number 2.

$$IF = (NI * NO)^2 \quad (1)$$

$$WIF = (NI * NO * Weight)^2 \quad (2)$$

Where in weight in Eq. 2 based on table 3

For large systems that use several changes made, a summation used to measure all business rules in the system, as shown in eq. Number 3 and eq. number 4:

$$IFA = \sum_{1=1}^{n} IF \quad (3)$$

$$WIF = \sum_{1=1}^{n} WIF \quad (4)$$

n: is the number of activity used in the system.

Information Flow for a Level (IF4L): IF4L metrics is an adaptation of a level of maintenance in a system that consists of three levels, which are: input text level, user interface level and output level, and they calculated as follows:

The Level Of Input Text (LOIT) defined as square the result of the Number Of Input Text (NOIT) by its weight (W); as shown as follows in Eq. Number 5.

$$LOIT = (NOIT * W)^2 \quad (5)$$

The Level Of User Interface (LOUI) defined as, square the result of the Number Of User Interface (NOUI) by its weight (W); as shown in eq. 6.

$$LOUI = (NOUI * W)^2 \quad (6)$$

The Level Of Output (LOO) defined as square the result of the Number Of Output (NOO) and its weight (W); as shown in eq. 7.

$$LOO = (NOO * W)^2 \quad (7)$$

The weight (W) of each level $=2^n$ Table 3 represents the corresponding n for each level.

Table 3: Weighted Levels

category	Activity	Weight
Enhancive	Replacing functionality	1
	Adding functionality	2
Corrective	Fix exciting functionality	1
	Improve conformance functionality	2
	Check existing test	1
	Adding a new test	2
Reductive	Limits existing functionality	1
	Remove existing functionality	2

Now, to calculate the IF4L for any software system, the software system will have several levels from the previous levels for each, so we have to use the following eq. 8:

$$IF4L = LOIT + LOUI + LOO \quad (8)$$

For the large systems which use several maintenance activities, each has a specific level; a summation of the measures of all levels of maintenance activities contained in the system obtained as shown in the following eq. 9:

$$IF4AL = \sum_{1=1}^{n} IF4L \qquad (9)$$

CONCLUSION

In this paper, we study the failure and success of software maintenance and emerging software quality models to reduce the failure software. Hence, we discussed the software quality models for specified maintenance purposes. This paper compares the quality model factors from maintenance aspects. Furthermore goes behind the definitions of the maintenance requirements form the software quality factors, sub-factors and criteria that affect the software failure and success.

Furthermore, new factors proposed to get clear and accurate differences between software quality models. This method requires to assign values for the sub-factors moreover the main factors, which is giving a clear picture of the differences between the models.

The values in this study were given equivalently between the factors and between the sub-factors that is because this comparison was generally. In a specific domain, the costs for each factor and sub-factors have to defined according to the selected domain. Finally, we proposed equations to compute the complexity of maintenance according to activities levels.

REFERENCES

[1]. V. Basili, P. Donzelli, and S. Asgari. A unified model of dependability: Capturing dependability in context. IEEE Software, 21(6):19-25, 2004.

[2]. B. W. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G. J.Macleod, and M. J. Merrit. Characteristics of Software Quality. North-Holland, 1978.

[5]. Wolski, M., Walter, B., Kupiński, S., & Chojnacki, J. (2018). Software quality model for a research-driven organisation—An experience report. Journal of Software: Evolution and Process, 30(5), e1911 .

[6]. Youness, B., Abdelaziz, M., Habib, B., & Hicham, M. (2013). Comparative Study of Software Quality Models. IJCSI International Journal of Computer Science Issues, 10(6), 1694-0814 .

[7]. Al-Badareen, A. B., Selamat, M. H., Jabar, M. A., Din, J., & Turaev, S. (2011). Software quality models: A comparative study. Paper presented at the International Conference on Software Engineering and Computer Systems.

[8]. Lee, M.-C. (2014). Software Quality Factors and Software Quality Metrics to Enhance Software Quality Assurance. British Journal of Applied Science & Technology, 4, 21 .

[9]. McCall, J. A., Richards, P. K., & Walters, G. F. (1977). Factors in software quality. Volume i. concepts and definitions of software quality.

[10] Miguel, J .P., Mauricio, D., & Rodríguez, G. (2014). A review of software quality models for the evaluation of software products. arXiv preprint arXiv:1412.2977 .

[11]. Boehm, B. W., Brown, J. R., & Kaspar, H. (1978). Characteristics of software quality .

[12]. Dreheeb, A. E., Basir, N., & Fabil, N. (2016). Comparative Study of Quality Models. International Journal of Computer Science and Electronics Engineering (IJCSEE), 4(1).

[13]. Dromey, R. G. (1995). A model for software product quality .IEEE Transactions on software engineering, 21(2), 146-162 .

[14]. Febrero, F., Calero, C., & Moraga, M. Á. (2016). Software reliability modelling based on ISO/IEC SQuaRE. Information and Software Technology, 70, 18-29 .

[15]. Esaki, K., Azuma, M., & Komiyama, T. (2012). Introduction of quality requirement and evaluation based on ISO/IEC square series of standard. Paper presented at the International Conference on Trustworthy Computing and Services.

[16]. Priyadarshini Tripathy and Kshirasgar Nail "Software Evolution and Maintenance a Practitioner's Approach", Wiley 2015.

[17]. Chapin N., Hale J. F., Khan K. M., Ramil J. F., and Tan W. G. 2001. "Types of software evolution and software maintenance". Journal of Software Maintenance and Evolution: Research and Practice, 13, 3–30.

The Zarqa University Deanship of Research funds this research