The Impact of Software Quality on Maintenance Process

Anas Bassam AL-Badareen, Mohd Hasan Selamat, 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. Software maintenance is considered as one of the main issues in software development life cycle that is required efforts and resources more than other phase. Studies estimated that the cost of software maintenance rapidly increased that reached the 90% of the total cost of software development life cycle. Therefore, it is considered as an economic impact in information system community. Several researches are intended to estimate and reduce the cost of this task.

This study introduces a model of software maintenance process that emphasizes the impact of the software quality on the maintenance process. The study presents the process of the software maintenance, and then discussed the quality characteristics that affect these tasks. Furthermore, the evaluation criteria for these factors are discussed.

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

I. Introduction

In order to keep the software useful and dynamic with the world changes, it has to be changed accordingly [1-2]. As the information technology industry gains maturity, the number of software systems having moved into maintenance is rapidly growing [3]. Software maintenance is a hard (and costly and error-prone) process in software life cycle [4-6]. The resources spent in software maintenance are more than that spent in other tasks of software development processes [7-8]. Several models were proposed to estimate software maintenance effort and cost has not yet been satisfactorily addressed [9]. Therefore, the cost of software maintenance is rapidly increased.

Souza [1] and Burch and Hsiang-Jui [10] show that the cost estimated for software maintenance is very high (80-90%) of the total cost of Software Development Life Cycle (SDLC). According to IEEE, the annual cost of the software maintenance in United States exceeds $70 billion [11]. Jones [12] states that, the cost of software maintenance has been increased in united states from 52% in 1995 to 76% in 2005 and is expected to increase steadily. Therefore, software maintenance is recognized as an economic impact in the information system community.

Effort estimation is the most relevant problem in the process of software maintenance. That it is a complex process by many aspects of software that affect maintenance activity [13].

This study aims to analyze the maintainability characteristics, classify the evaluation criteria, 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.

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 the software products 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 [3].

Early of sixties, the size and the complexity of the software started rapidly grow. More attention has been given to the concept of software evaluation in the development and maintenance processes. The evaluation started based on code size and error correction. Nowadays, the evaluation is an essential in software systems usage [14].

Software maintenance is differing from physical maintenance that is the software does not physically wear and it can be delivered with undiscovered flaw [15]. Sommerville [16] estimated that, nearly 250 billion lines of source code were being maintained by organizations in 2000, and this number is growing.

Tiako [14] 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

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

Mohd Hasan Selamat is a professor of software engineering with University Putra Malaysia. Phone: 603-89471720; Fax: 603-8946 6577; e-mail: hasan@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.

specification and the implementation of the software; and (g) the changes of strategic needs.

IEEE Standard 1998 for Software Maintenance describes the process for managing and executing software maintenance activities [17]. IEEE standard 2006 [18] defines the activities and tasks of software maintenance, and provides maintenance planning requirements.

III. Maintenance Process

The modification is become a complex and costly process, that is large and complex software system could be delivered with undiscovered flow. Therefore, several tasks are considered in order to achieve this process. Software maintenance as shows in figure 1 consists of four main tasks, understand, analyze, modify, and test the intended system [19].

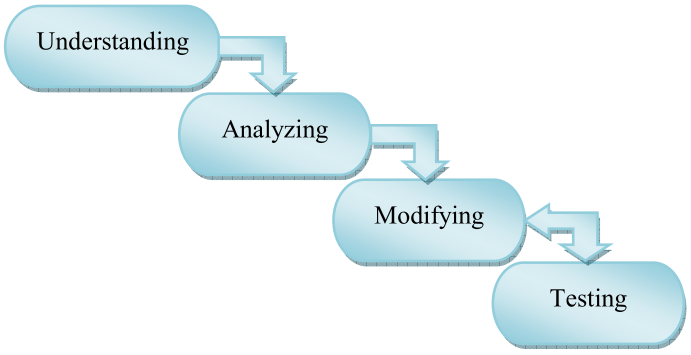

Figure 1: Software Maintenance Processes

System understanding: is the process of realize the system functions and their relationships. It is considered as an essential base of inspect and modify a software product.

System analysis: is very important task that concerned on identifying the required modification to correct, enhance, or adopt the system.

System modification: is the task of changes and corrects the inspected functions within the system.

System testing: evaluates whether the modification that has been made achieves the maintenance goals. This test is differing from normal test, which considered only the modification that has been made during modification task and the side effects on other functions. Thayer [20], defined it as a functional test is called regression test, which is used to determine whether the modification has altered software functions that were to remain changed.

IV. Software Maintainability

Since there is no clear definition of measuring software maintainability, from different points of view the maintainability measurement has 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, that it bypass the developers the cost of build of new system in order to replace the existing [21-22]. Mittal [23] 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 [24] 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 [25], considers the maintainability as integrated measure of three factors, Readability of Source Code (RSC), Documentation Quality (DQ), and Understandability of Software (UOS).

V. Maintainability Factors

In order to evaluate the maintainability of the software, the measurement is considered based on it is characteristics. These characteristics are classified based on the maintenance processes: understanding, analyzing, modifying, and testing. The characteristics required for each process are derived and concluded from the most well-known software quality models and literature.

A. Understandability

If you can understand your system, you can change it effectively [26]. The first task of software maintenance process is to understand the existing software [27]. One of the major concerns of any maintenance organization is to understand and estimate the cost of maintain released software systems [28]. 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 phase, which 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 required to be modified.

Software understandability is a very important quality factor allows the developers to understand the structure of the system easily, which simplifies the processes of the maintenance [29]. Therefore, system understanding depends on cognitive abilities and preferences, familiarity with the application domain, and the set of support facilities that provided by the software engineering environment. Thus, understandability affects the total effort and cost required in maintenance process.

Therefore, software understanding aims to present enough information about the system. This information is written by software engineers and programmers during software development lifecycle [30]. If the documentation and the source code are not correlated, the maintenance process will be very difficult and not accurate [25]. 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 software characteristics are considered and intended to be measured. Figure 1 shows the characteristics related to software simplicity and the information used to express the software functions and their relations.

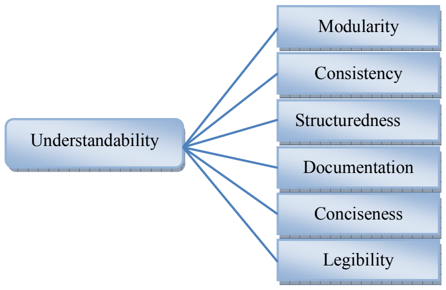

Fig. 1 Understandability Factors

B. *Analyzability*

Analyze a developed software is an essential to characterize the software behavior [31]. In order to identify the errors or the parts required to be modified, system analysis is required. 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 a modification which may be in primary concern [32-33]. Figure 2 presents the characteristics that affect software analyzing.

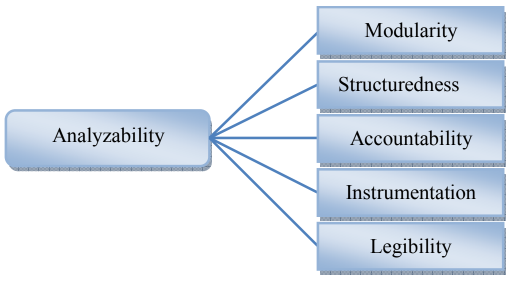

Fig. 2 Analyzability Factors

C. *Modifiability*

One of the greatest challenges facing software engineers is manage the change control. It has been estimated that the cost of change control can be between 40% and 70% of the life cycle costs [15]. 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 changed, 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 side 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.

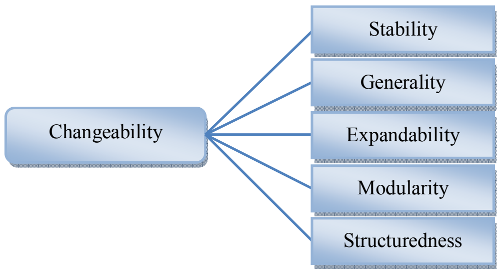

Fig. 3 Changeability Factors

D. *Testability*

Software testability is defined as “The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met” [34]. Software testability is the ability to test whether the software meets its requirements. It is used to inspect a software defects early before release a software product [35]. According to Singh [36], software testability has been defined based on the controllability and observability. Controllability represents the ability to control the input, whereas the observability presents the ability to measure the output of the system.

In maintenance process, testability presents whether the modification has achieved its goals and if there is any side effects on other functions. 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 the required functions. Figure 4 shows the software characteristics that help in this process.

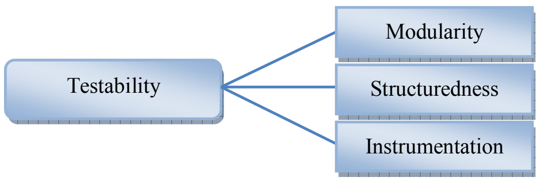

Fig. 4 Testability Factors

VI. Maintainability Criteria

A. *Documentation*

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 their relationships. 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.

System documentation 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. The documentation is a relevant even if it is not up to date [37]. However, 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.

Forward [37], conducted a survey of professionals in software industry. According to the result of the survey, he observed that, the documentation is an important tool for the communication and it is content can be relevant even if it is not up to date. Lehner [38] proposed a decomposition of software documentation quality attributes that proposed by Arthur/Stevens. In addition to, eight methods that have been used to measure documentation quality.

Software documentation can be measured by several characteristics that can present whether the documents are useful to describe a system components and structure. The main characteristics of the software documentation are defined as follow:

Clarity: documentation clarity can be measured by calculate the inverse of the document ambiguity. The document ambiguity is measured by ratio of the ambiguous statements to the total statements in the document.

Consistency: the consistency evaluates the contradiction in the software documentation, whether any oppositions between two or more statements within a document.

Completeness: the completeness evaluates whether the documentation covers all of the software components and their relations.

Conciseness: the conciseness presents the ratio of the nonrelated or non useful statements to the total statements in the documentation.

B. Conciseness

It is a code characteristic means that excessive information is not present [39]. 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. According to Thayer [20] , the conciseness is a software characteristic provides a function implementation with a minimum amount of code.

Conciseness measurement is considered at statement level, which presents the ratio of the concise code statement to the total number of statement on the software.

C. Consistency

According to Boehm [39] the consistency is presented from two points of view, internal and external consistency. Internal consistency is a code characteristic evaluates whether it contains uniform notation, terminology and symbology within itself. External consistency presents whether the code content is traceable to the requirements.

The consistency is important for correct interpretation of 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. Furthermore, the system consistency can be achieved by covering the consistency in each phase of software development and between different phases of software development.

Internal consistency considered the code consistency at three main levels, statement level, function level, and class level. Statement level presents the ratio of the consistent statements within a function, which is equals to the ratio of the consistent statement to the total number of the statements within a function. The function level presents the ratio of the consistent function to the total number of functions within a class. The class level presents the ratio of the consistent class to the total number of classes within a system.

External consistency, at this level the code is considered as a part of the software development phases. Therefore, the consistency is evaluated based on the requirement definition through development phases and resulted on the source code. The consistency presents the sequential smooth of the requirements through development phases.

The evaluation traces every requirement defined early in the requirement elicitation through software development life cycle till reach the source code. Therefore, the evaluation calculates the consistency of the function in every level of development and then the ratio of the consistency to the total number of the functions within a system is measured.

D. Legibility

Clear design is good design [26]. It is a code characteristic presents whether the easily of the function to be discerned by reading the code [39]. Legibility of the system simplifies the job of identifying the required modification, which defined that the information should be easy to read.

The legibility of the class presented the ration of the legibility functions to total number of functions. The legibility of the function is the ratio of the legible statements to the total number of statements within a function.

E. Accountability

The need for accountability has steadily grown, since the computer system start growing in 1970s [40]. It has become a major concern for businesses around the world [41]. Therefore, it has been considered as a goal for software quality assurance [42].

According to Lin [41], Schedler [43] provides a definition that succinctly captures the essence of accountability in real life:"A is accountable to B when A is obliged to inform B about A's (past or future) actions and decisions, or justify them and to be punished in the case of misconduct".

The accountability tracks every action occurred in the system [44], which allows to identify the errors and it is causes efficiently. Moreover, it decrees the relationships between the components that reduced the scope affected by error, which reduce the area of the inspection.

According to Boehm [39] the accountability is a code characteristic presents whether the code usage can be measured. The accountability can be calculated at function level, it presents the ratio of accounted functions to the total number of the functions within a class.

F. Modularity

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, which allow understanding a system part by part instead of understanding the system as a one part. Moreover, it allows modifying some parts of a system independently from other parts [45]. Therefore, it is considered as a key way of innovation and mass customization. Common metrics of design modularity include coupling, cohesion, and separation of concerns [45].

The modularity simplifies software analysis by decomposing the system into several manageable parts. This allows identifying the errors or the functions that required a modification easily and efficiently. 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 errors inspections and the reasons caused these errors. The modular software allows each part of the system working independently with minimum effects on the other parts of the system.

Basically, modularity is considered early in requirement elicitation phase, it can be presented in a use case model. The ability to consider every use case as an independent system within software development phases is the essential point in modularity. In terms of source code, the modularity presents whether the system is divided into several manageable classes and the relationships between these classes.

Cai [46] discussed the issues in modularity, the current evaluation is only based on source code, while a design is not considered yet. Therefore, in this evaluation the modularity is considered based on requirement definition, which presented use cases into implementation phase presented by source code.

The essential base used in object oriented design is a use case model, which it used to classify the users' requirements into manageable groups intended to achieve same goals. Therefore, the use case is considered as a base to evaluate system modularity.

G. Structuredness

A good structure is an important software quality aspect [47]. It is a code characteristic presents a definite pattern of organization of its interdependent parts [39]. 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 structuredness considers the coupling between software parts and the effects of software analyzability by identifying the scope of the error inspection process. Software structuredness is very important in terms of modification scope, which considers the coupling of the software components. The reducing of the relationships between the components within the system considered in structuredness factor. It is important to reduce the efforts and the complexity required to test the software components and their relations.

H. Instrumentation

As software increasingly grow and complex, supporting tools are became an essential and crucial in software life cycle. Whereas, the high cost of software maintenance, it is essential need for automated tools, in order to analyze and understand a large and complex software system [27]. These tools support the construction, maintenance and analysis [48]. The instrumentations simplify the task and reduce the time and the efforts. Moreover, the instrumentation increases the accuracy of the software analyzing. The instrumentations used on the testing increase the accuracy of errors inspections and decrease the efforts and the time required to test the system.

I. Stability

Normally, the modification of parts of the system caused a change of the component behavior. This modification may cause a side effects on other parts, which is mean solve a problem by other problem. Therefore, the concept of stability considered to reduce the impact of software modifications by dividing a system into stable and unstable models [49].

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.

J. Generality

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 [50].

K. Expandability

It is a characteristic of the code that presents the ability to accommodate any expansion in the software functions or storages [39]. 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. According to Thayer [20] expandability is the degree of the effort required to enhance or modify a software functions.

VII. Discussion and Implication

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 software factor has been added. This factor presents whether those processes are covered by any international standard or certificate. The compliance can be considered for the whole processes or for each one of them independently. Figure 6, shows the structure of the maintainability characteristics.

Modularity and structuredness are the main characteristics required in software maintenance, which used in all of the software maintenance tasks. These characteristics considered on the divide a system into several manageable parts communicate with each other. Therefore, the maintenance effort is reduce as much as system decomposed instead considered the whole of the system in such modification, whereas in the structuredness shows the effects of the modified part on the other parts in the system.

The documentation is an essential archived software documents that have been conducted through software development. These documentations are very important in order to understand the functions and their relations in the system.

The legibility is considered as a base of software maintenance, that is this characteristic presents the ability to read and understand the system and to identify the part required to be modify. Whereas, the maintenance is not a direct continues with other phases of software development. The maintainers have to go back to the archive, which required reading and understanding a system from a beginning. Both of source code and software documentation are required to be eligible.

The consistency is considered in both of software development process and software product. The consistency of the process considered the smooth flow of the development process that it can be presented in the documentation. The

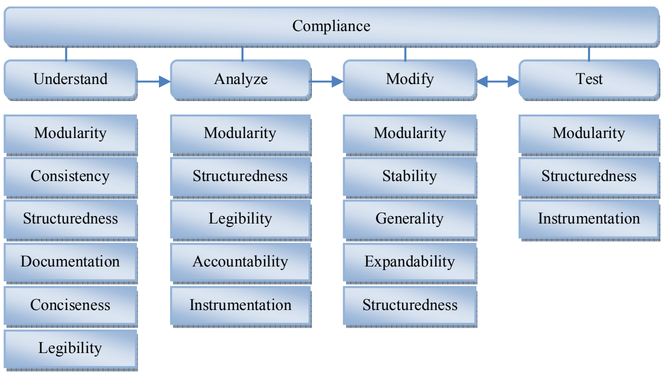

Figure 5: Software Maintenance Framework

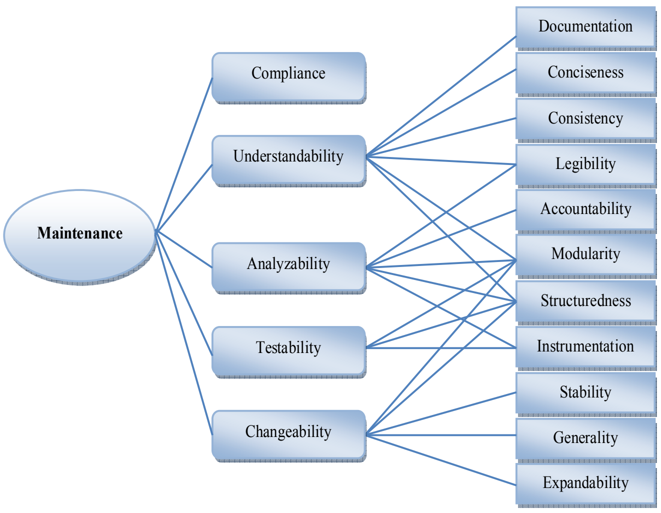

Figure 6: Software Maintainability Characteristics

consistency of the software product considered the free of the contradictions between the software functions.

The conciseness presents whether both of the source code comments and software documentation includes only necessary and useful information and free of any nonsense information, that may cause confusing and misunderstanding for software developers.

The accountability is a security characteristic used to trackback every action occurred in the system during it is operation. This characteristic is considered in software maintenance in order to identify and clarify the failures that occurred during system use. In this technique, the maintainer rollback to the last action occurred before the failures, which is very helpful to identify what are the reasons of the failures and where it was happened.

The development instrumentations were proposed when the software size and complexity start growing. These instrumentations are used to analyze and test the large and complex system efficiently. The major advantages of the development tools that save the time required to perform a development task and reduce the ratio of the errors that may caused by human.

The stability characteristic is used to presents whether the modifications affect other functions in the system. It is a clear on object oriented software, which the modified component may affect the components that rely on it.

Expandability presents the ability to modify and extend a software functions efficiently. The generality is a very important characteristic for both maintenance and reusability. The generality shows the flexibility of the software to manage different types of data that may required.

VIII. Conclusion

In last decades, the growing of the software size and complexity caused serious problems in maintenance problems. Several studies were proposed different solutions from different points of view, in order to overcome this issue. 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. Moreover, the evaluation criteria of these characteristics were discussed. Furthermore, in order to evaluate whether the maintenance process or some of its task follows any standard or international certificate, the compliance factor was considered.

The proposed framework is a theoretical framework, which lack of practical evaluation. Therefore, this issue is considered in future in order to be validated.

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] 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.

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

[4] 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.

[5] M. KERNAHAN, *et al.*, "Extracting Traceability Information from C# Projects," in *WSEAS International Conference on ENGINEERING EDUCATION*, Athens, Greece, 2005.

[6] 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.

[7] J. Olivas, *et al.*, "Using fuzzy prototypes for software engineering measurement and prediction," 2002.

[8] F. García, *et al.*, "A Tool for the Management of the Software Maintenance Process," in *5th WSES International Conference on Circuits, Systems, Communications and Computers*, Rethymno, Greece, 2001.

[9] H.-W. Jung and D. R. Goldenson, "Evaluating the relationship between process improvement and schedule deviation in software maintenance," *Information and Software Technology,* vol. 51, pp. 351-361, 2009.

[10] 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.

[11] D. V. Edelstein, "Report on the IEEE STD 1219-1993-standard for software maintenance," *SIGSOFT Softw. Eng. Notes,* vol. 18, pp. 94-95, 1993.

[12] C. Jones, "The Economics of Software Maintenance in the Twenty First Century, 2006," *Ryan North and James Choi: Leveraging Software Performance Engineering to Enhance the Maintenance Process,* vol. 68, 2006.

[13] A. D. Lucia, *et al.*, "Effort estimation for corrective software maintenance," presented at the Proceedings of the 14th international conference on Software engineering and knowledge engineering, Ischia, Italy, 2002.

[14] 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.

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

[16] I. Sommerville, *Software Engineering*, 6th ed.: Addison-Wesley: Reading MA,, 2000.

[17] "IEEE Standard 1219," in *Standard for Software Maintenance*, ed: IEEE Computer Society Press, 1998.

[18] "IEEE Std 14764-2006.," in *IEEE Standard for Software Maintenance*, ed: IEEE Computer Society Press, 2006.

[19] A. ECONOMIDES, "Evaluation of Collaborative Learning Systems," 2005, pp. 169-175.

[20] R. H. Thayer, "Software Maintenance," *Software, IEEE,* vol. 22, pp. 103-103, 2005.

[21] 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.

[22] 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.

[23] H. Mittal and P. Bhatia, "Software

maintainability assessment based on fuzzy logic technique," *SIGSOFT Softw. Eng. Notes,* vol. 34, pp. 1-5, 2009.

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

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

[26] M. Feathers, "Before clarity [software design]," *Software, IEEE,* vol. 21, pp. 86-88, 2004.

[27] E. Hill, *et al.*, "Exploring the neighborhood with dora to expedite software maintenance," presented at the Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, Atlanta, Georgia, USA, 2007.

[28] 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.

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

[30] 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.

[31] M. M. Diep, "Analysis of a deployed software," presented at the The 6th Joint Meeting on European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering: companion papers, Dubrovnik, Croatia, 2007.

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

[33] 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.

[34] "IEEE Standard Glossary of Software Engineering Terminology," *IEEE Std 610.12-1990,* p. 1, 1990.

[35] J. Fu, *et al.*, "Present and future of software testability analysis," in *Computer Application and System Modeling (ICCASM), 2010 International Conference on*, 2010, pp. V15-279-V15-284.

[36] Y. Singh and A. Saha, "Improving the testability of object oriented software through software contracts," *SIGSOFT Softw. Eng. Notes,* vol. 35, pp. 1-4, 2010.

[37] A. Forward, "Software Documentation–Building and Maintaining Artefacts of Communication," Citeseer, 2002.

[38] F. Lehner, "Quality control in software documentation:: Measurement of text comprehensibility," *Information & Management,* vol. 25, pp. 133-146, 1993.

[39] B. Boehm, *et al.*, "Quantitative evaluation of software quality," 1976, p. 605.

[40] P. Malone and B. Jennings, "Distributed support for public and private accountability in digital ecosystems," presented at the Proceedings of the International Conference on Management of Emergent Digital EcoSystems, France, 2009.

[41] K. J. Lin, *et al.*, "Accountability Computing for E-society," in *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, 2010, pp. 34-41.

[42] S. Eriksén, "Designing for accountability," presented at the Proceedings of the second Nordic conference on Human-computer interaction, Aarhus, Denmark, 2002.

[43] A. Schedler, *et al.*, *The self-restraining state: power and accountability in new democracies*: Lynne Rienner Pub, 1999.

[44] 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.

[45] C. Yuangfang and S. Huynh, "An Evolution Model for Software Modularity Assessment," in *Software Quality, 2007. WoSQ'07: ICSE Workshops 2007. Fifth International Workshop on*, 2007, pp. 3-3.

[46] Y. Cai and S. Huynh, "Measuring Software Design Modularity," 2008.

[47] M. H. Ammann and R. D. Cameron, "Measuring program structure with inter-module metrics," in *Computer Software and Applications Conference, 1994. COMPSAC 94. Proceedings., Eighteenth Annual International*, 1994, pp. 139-144.

[48] A. R. Dalton and J. O. Hallstrom, "nAIT: A source analysis and instrumentation framework for nesC," *Journal of Systems and Software,* vol. 82, pp. 1057-1072, 2009.

[49] C. Chia-Chu, "Software Stability in Software Reengineering," in *Information Reuse and Integration, 2007. IRI 2007. IEEE International Conference on*, 2007, pp. 719-723.

[50] A. B. AL-Badareen, *et al.*, "Reusable Software Components Framework," in *European Conference of Computer Science (ECCS '10)*, Puerto De La Cruz, Tenerife, 2010, pp. 126-130.