Relationship between Factors of Quality Models and the System Development Life Cycle

Basit Habib
Bahauddin Zakariya University Multan.

Rana Aamir Raza Ashfaq
Bahauddin Zakariya University Multan.

ABSTRACT

Gaining quality of software depends on the way how it is developed. To develop a quality software "System development life cycle" is the best technique to be adopted. Good software has the ability to map itself on a Quality Model so that its credibility can be seen on a set of factors along with their criterion. In this paper a Relation between phases of SDLC and popular Factors of different Quality Models is shown.

Categories and Subject Descriptors

1.Why certain factors are there against certain phases of SDLC?

2. Why certain phases of SDLC there against certain factors? are discussed.

General Terms

Reliability, Experimentation, Performance, Design,.

Keywords

Quality, Factors, Criterion, System Development Life Cycle, Quality Models, Phases.

1. INTRODUCTION

1.1 What is a Software Quality Model?

A software quality model is a set of Factors and Criterion against those factors. The main idea of a software quality model is to show such attributes which can make a software work properly in all manners of its Domain of work. A Quality model is based a set of factors and these factors are based on a set of different criterions. To understand a brief but deep knowledge of a quality model, its definition needs to be elaborated [1],[4].

1.2 What is a Factor?

In a quality model the factor or set of factors (as a quality model is based on a set of factors) are similar to the bones of a quality model which develop a skeleton structure from head to toe which shows the positioning of steps of a project or work to be done. If the steps are properly merged into the bone then the skeleton becomes more and more strong. But the question is how to make this skeleton move in a proper manner that every bone of the skeleton shows full and exact working?

For the best to be gained by the skeleton we need to again look at the second half of the definition of a quality mode [6]l.

1.3 What is criterion?

Criterion is the sub sets of a factor. They are the joints of the bones (factors) which show how the step of a project or work can show flexibility for the bones to move in a free motion and show movement for the skeleton can move. The criterion is important as they strengthen the skeleton of a quality model.

Criterion basically shows more domains on which elaborately the problem or step can be distributed for their better development or solving solution.

So the Factors and Criterion is the base of any quality model and by using them the quality of a product can be maximally achieved [6].

2. History of Quality Models:

The idea of quality of software product was initially introduce by McCall *et al.* (1977). The idea soon gathered widespread acceptability and several other authors contributed significantly in the exploration and refinement of the idea. A chronological order will be observed in the following discussion. We will restrict ourselves to the literature related to following four models as Software Quality Engineering Society recognizes only those[1],[4],[6].

- McCall's Quality Model.
- Boehm's Quality Model.
- Dromey's Quality Model.
- ISO/IEC 9126 Quality Model.

2.2 Explanation of Quality Models:

2.1.1 McCall's Quality Model

In a technical report McCall *et al.* (1977) introduced a hierarchical definition of factors affecting software quality. The definition was comprehensive to cover wide range of software development phases and was able to split software oriented and non-oriented characteristics. Programming language-independent metrics were developed for software-oriented factors using Air Force databases. The report was prepared for Rome Air Development Centre (ISIS) and the three authors belonged to General Electric Company. Although the objective of the study and the report was to establish a concept of software quality and to provide the host organization with a mechanism to quantitatively specify and measure the desired level of quality in a software product in terms of software metrics but the idea was welcomed beyond the host organization.

The report is divided into three volumes namely: Concept and Definitions of Software Quality; Metric Data Collection and Validation; and Preliminary Handbook on Software Quality. This model is considered as the most influential one. This may be because being the earliest and classical it is the mother of the all models. It defines the software product as hierarchy of quality factors, quality characteristics and quality metrics. The model defines a set of eleven quality factors which could, by associating one or more metrics to each factor, be used to gauge quality of software product fully. It not only defines metrics for each criterion and a normalization function which establishes and validates a relation between *the metrics for all of the criteria of a factor* and *an overall rating to that factor* but translates the results into guidelines[6],[9].

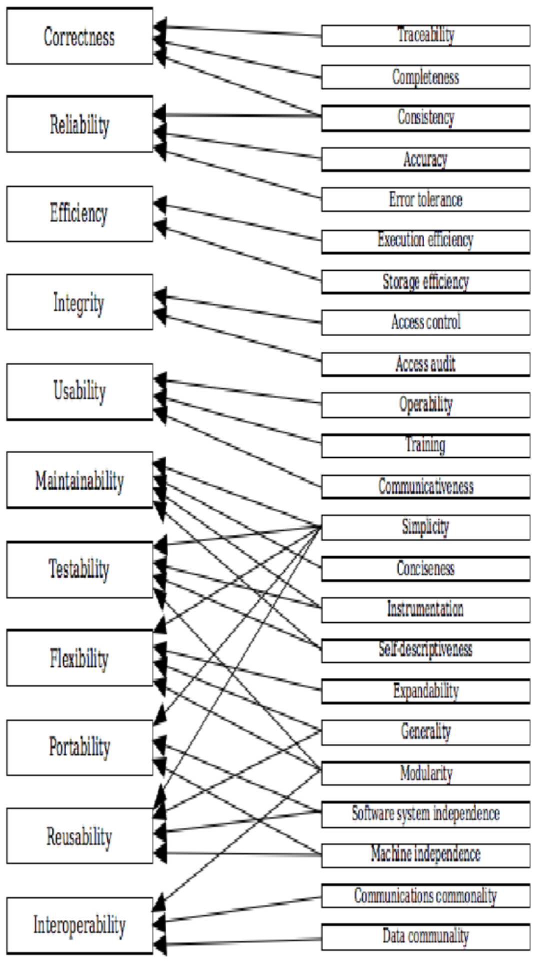

Figure 1. McCall's Quality Model Adapted from McCall (1977) and Pfleeger(2003)

2.1.2 Boehm's Quality Model

The wide spread popularity of McCall's model attracted the attention of the readers and writers. Within a year of its inception McCall's models received appreciation and criticism. The most prominent material in this regard was presented by Boehem *et al.* (1978) who had started presenting such work a year before McCall's model as Boehm (1976).

Boehm's model defines the quality of software in quantitative terms by means of set of predefined attributes and metrics. He defined three-level hierarchy namely high, intermediate and primitive with tree, seven and twenty-three (fifteen distinct) characteristics respectively. These characteristics collectively contribute to the overall quality level [6].

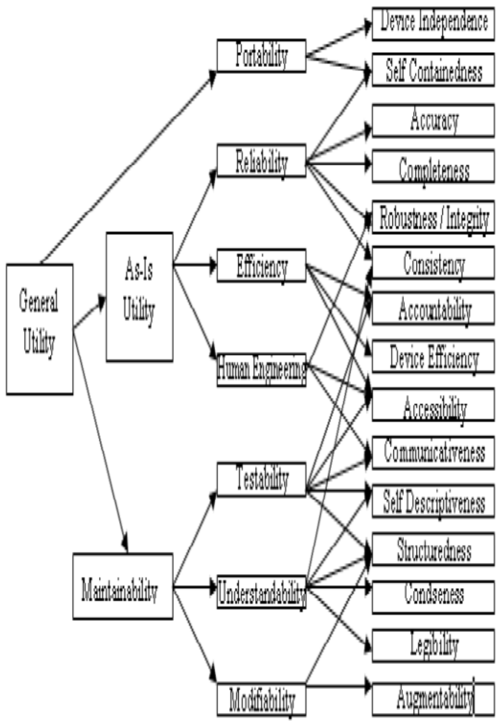

Figure 2. Boehm's Quality Model Adapted from Boehm(1978) and Pfleeger(2003)

The characteristics at the highest level, for example, can be explained as below:

As-is utility which determines how to use quality on the basis of as-is and where-is basis.

Maintainability gauges the level of ease in maintaining software, if required.

Portability means if software is workable in a different environment.

2.1.3 Dromey's Quality Model

Although Dromey termed it as Framework not the model but it is recognized as so. Dromey's (1995) model takes a different way to software quality than the two models defined previously. Dromey (1995)states:

"What must be recognized in any attempt to build a quality model is that software does not directly manifest quality attributes. Instead it exhibits product characteristic that imply or contribute to quality attributes and other characteristics (product defects) that detract from the quality attributes of a product. Most models of software quality fail to deal with the product characteristics side of the problem adequately and they also fail to make the direct links between quality attributes and corresponding product characteristics."

That is, Dromey presented a different type of model which emphasizes that it is impossible to build high-level quality attributes like reliability or maintainability into a product, rather developers may build properties that are helpful in achieving the quality targets. The distinguishes the model from other models by verifying that it allows the quality required to be achieved but the successful application of the Dromey's model requires that the various groups involved in the development of a software product must agree on what quality attributes should be achieved and to what level. The Dromey's product model can be depicted as Fig. 2.3

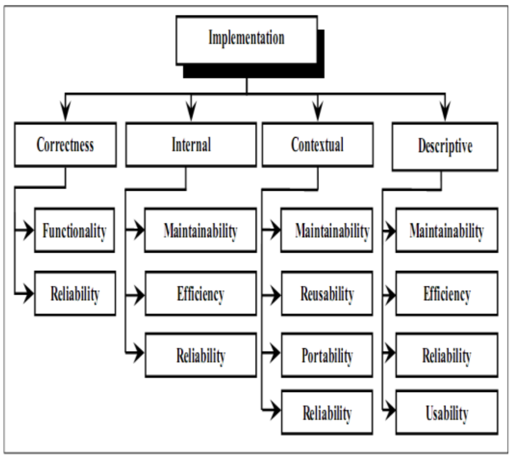

Figure 3. Dromy's Quality Model Adapted from Dromey(1996) and Pfleeger(2003)

This model received some criticism as well as it states that the high level characteristics of quality will manifest themselves if the components of the software product— from the individual requirements to the programming language variables—exhibit quality-carrying properties. The provision of quality components does not guarantee quality of the ultimate product. Just as quality of the individual ingredients of an apple pie does not guarantee quality of the apple pie unless you are experienced baker [3].

2.1.4 ISO/IEC 9126 Standard

ISO/IEC 9126 is an international standard, given by International Standard Organization (ISO) and International Electrotechnical Engineering to manage quality of software products including those software products whose failure may be detrimental to lives. It provides an all-inclusive specification and evaluation model for the quality of software products[6].

It has been divided into four parts as:

Quality Model (ISO/IEC TR9126-1 dated 21-06-2001)

It classifies software quality in a structured set of characteristics some of which are part of other standard quality models as quality factors.

Table 1. Quality Factors Present in Various Quality Models

Characteristics↓ Model→	McCall's	Boehm's	Dromey's
Functionality			✓
Reliability	✓	✓	✓
Usability	✓		
Efficiency	✓	✓	
Maintainability	✓	✓	✓
Portability	✓	✓	

External metrics (ISO/IEC TR9126-2 dated 09-07-2003)

These metrics are used to measure the characteristics (and sub-characteristics) listed in the quality model presented above. These are applicable to running software.

Internal metrics (ISO/IEC TR9126-3 dated 09-07-2003)

These metrics are used to measure the characteristics (and sub-characteristics) listed in the quality model presented above. These are applicable to static software.

Quality in use metrics (ISO/IEC TR9126-4 dated 07-03-2004)

It identifies the metrics used to measure the effects of the combined quality characteristics for the user.

3. System Development Life Cycle

SDLC the abbreviation of "System Development Life Cycle" as it is basically a technique which is used to develop any kind of a system. Now it depends that which type of system can be the one to be developed. The system can be any software system, any system based on the developing on any business strategy, and it also is concerned with the developing of any other automated system.

The SDLC is based on seven different phases which are all dependent on each other and show a relevancy between them. By seeing this relevancy between them, they also show the ability to recursively call each other if needed for the betterment of the system while being developed.

In software engineering, the phases of SDLC are very much similar to the phases of a "Water Fall Model". As they perform the similar working of:

- Writing the code.
- Fix and removal of errors.

As seen that Preliminary investigation, Requirement Specification, System analysis and design, are used for the developing or in other words used for writing the code of any project. While the phase of Integration and testing can be declared as a neutral or a phase which is used for determining the capacity of the project. And Installation and acceptance along with maintenance is used for the fix and removal of errors in any project.

So these are the seven main phases which map on the above mentioned two clauses to develop a tolerance free project.

Relation between Phases of SDLC and Software Quality Factors

The table 2 summarizes the relation of various phases of System Development Life Cycle to various Factors suggested in four recognized software quality models proposed by McCall, Boehm, Dromey, and ISO/IEC 9126.

The Table 2 can be analysed in two different ways. Mainly the table has the potential to answer following questions.

4. Why certain factors are there against certain phases of SDLC?

Why certain phases of SDLC there against certain factors?

Why Certain Factors are There against Certain Phases of SDLC?

To answer first question we will be required to each row and for answering the other question each column will be discussed. First question is answered first and each phase of SDLC is taken sequentially [5].

Table 2. Relation Between Phases of SDLC and Quality Factors

Quality Factors→ Phases of SDLC ↓	Correctness	Efficiency	Flexibility	Functionality	Human Engineering	Integrity	Mantainability	Modifiability	Portability	Reliability	Testability	Understandability	Usuability	
Preliminary Investigation				✓				✓				✓		3
Requirement Specification				✓						✓			✓	3
System Analysis and Design		✓		✓	✓					✓		✓	✓	6
Integration/ Testing	✓	✓		✓						✓	✓			5
Implementation	✓	✓		✓		✓								4
Installation and Acceptance	✓			✓										2
Maintenance			✓				✓	✓	✓					4
	3	3	1	6	1	1	1	2	1	3	1	2	2	

Phase-I (Preliminary investigation)

The required factors are *understandability, modifiability,* and *functionality*. At one hand, system developers' understanding of what is required works as the foundation of the final product but on the other hand intender's understanding of what is going to be developed gives a go-ahead to the developers. At the preliminary stage of system development this factor will induce potentials for the high quality in the final product.

There are chances that the intentions of the intender could not be understood properly. Similarly explanations of the developers' may not get down to the intender at a satisfactory level. Because of this or other such reasons modification may become inevitable. The potential of modification improves functionality by taking the rigidity off the system. These abilities are thus required to be adjusted at the very initial stage of SDLC [8], [9].

Phase-II (Requirement Specification)

The title of the phase is elaborative enough where we translate project goals into defined functions and operations. End-users' information need is also analysed at this phase.

Three factors namely reliability, usability, and functionality are found relevant here.

Reliability which is the ability of the software not to go failed while running can only be managed if the requirements have been understood and transformed into functions appropriately. This will make the software work as per design and will make the system dependable.

The other factor found relevant to this phase of SDLC is usability. This factor says that the function performed by the program is also useful elsewhere and is robust against human error. To get the quality factor incorporated, efforts should be done at this stage of SDLC.

Functionality is the third factor involved here. At every stage, it must be assured that the software is fully functional in all of its areas of application. It can only be guaranteed if requirements have been taken and documented at a satisfactory level.

Phase-III (System Analysis and Design)

Desired features and operations of the software products are described in detail at this phase of SDLC. These details are presented by means of screen layouts, business rules, process diagrams, pseudo-code and other documentation.

This seems to be the most influential phase of SDLC as it covers a lot detail. This influence makes it attract six factors namely Efficiency, Functionality, Human Engineering, Reliability, Understandability and Usability. Before going ahead these factors deserve brief look at them.

Efficiency: Software should fulfil its purpose without waste of resources i.e. getting most of the utilized resources. It is usually gauged with respect to time and storage.

Functionality: The capability of the software product to adhere to standards, conventions, or regulations in laws and similar prescriptions relating to functionality

Human Engineering: it is about robustness, integrity, accessibility, and communicativeness of the involved humans [5].

Reliability: The reliability of the software assures that it can be expected to perform its intended functions satisfactorily.

Understandability: Understandability of the software states that the purpose of the software is clear. This implies that the variable names or symbols are used consistently, modules of code are self-descriptive, and control structure is simple or in accordance with a prescribed standard.

Usability: It means that it is a blend of three factors as the software should be reliable, efficient and human engineered. It implies that the function performed by the program is useful elsewhere, is robust against human error, or does not require excessive core memory.

The six factors associated with the phase-III are all related to development and allied activities. All six factors are desired quality features of the ultimate features. If those are accommodated at the initial stages, only then the desired quality can be expected in the final software.

Phase-IV (Integration/Testing)

Integration means combining two or more than two software modules together to grow a bigger module or the final software. Managing development of the whole project or larger modules is usually difficult. The development of manageably smaller modules and then their integration into a larger one not only facilitates the developer during development but also makes the testing of the modules easier. The testing may be required to be redone after integration as there are chances of compatibility issues among the modules being integrated. At the rerun only regression type testing would suffice.

Following factors are listed to be relevant at the stage [5], [6].

Correctness: In the perspective of integration, correctness is the ability to get various modules integrated correctly.

Functionality: Once integrated all the modules of software involved demonstrate full functionality and operation without any issues of compatibility or whatever.

Efficiency: Maintaining efficiency in relatively smaller modules is far easier than their larger counterparts. It is possible because of having more concentrated visualization for specialized modules.

Reliability: As it has already been discussed, modular approach provides better conception and visualization of the final product. It, thus, improves testing and consequently reliability of the software. So we could correctly expect that the software will perform its intended functions at a high level of satisfaction.

Testability: The modules and merger of modules to form the semi-final or final product should be testable. If a module(s) cannot be tested then it can never be reliable and may ultimately fail to perform the intended functions. This factor defines different methods and tools to test the part or whole of the software project being developed.

The five factors associated with the phase-IV are all related to integration and/or testing. Testability and reliability are obviously linked to testing whereas correctness, efficiency, and functionality have direct link with the integration. Three of the five factors— namely efficiency, functionality, and reliability— are same as discussed in phase-III of SDLC and are desired quality features of the ultimate features. Accommodation of these factors at the initial stage assures the desired quality in the final software.

Phase-V (Implementation)

No software will earn any appreciation or criticism of the end-users (the ultimate masses of users) unless it has gone implemented. The real run is not possible without implementation. This phase of SDLC is all about implement and following quality factors proposed in the recognized software quality models are relevant:

Correctness: In this perspective, correctness means assurance regarding correct implementation of the software in all areas of application.

Functionality: Once the software has been implemented it must be assured that it is fully functional in all of its areas of application.

Efficiency: The implementation phase will not be considered finished until it has been seen that the software is efficient in different dimensions such as execution time and memory usage. It should also be checked if change in working environment such as computing or physical environment should not deteriorate the efficiency.

Integrity: Access control is the basic tool for software security. It is desired that access may be given in an appropriate hierarchical order. Access beyond a defined level may not be available to users. Moreover log of (un)successful attempts for (un)authorized access may be prepared and notification of unusual activities may be reported to the system administrator.

Correctness, functionality, efficiency and integrity are the factors which have a direct and influential impact on the successful implementation of the software. Non incorporation of any or all of these factors makes us compromise on the success of the implemented software. This explains inclusion of these factors against the phase of SDLC [5].

Phase-VI-(Installation and Acceptance)

End-users' acceptance for the software is misconceived as final acceptance but the acceptance of the intender is final. The acceptance of software in this perspective depends on the smooth running and successful installation. The smooth running largely depends upon successful installation and compatibility with the platform and allied software being run on the platform concurrently. The level of satisfaction leading to acceptance should not get affected by the conventional resistance from the end-users. An objective type analysis is thus required to reach at the right decision on the final acceptance. Relevance of the following factors is meaningful here.

Correctness: In this perspective too, correctness means assurance regarding correct and successful implementation of the software in all areas of application. Once the software is correctly working it will have a higher probability of getting accepted.

Functionality: Once the software has been correctly installed it must be assured that it is fully functional in all of its areas of application. If the software is fully functional in all areas of application then it will be hard to get rejected.

Correctly installed and fully functional software is more likely to be accepted.

Phase-VII-(Maintenance)

To coop with the ever-changing scenarios of the dynamic world, the implemented software is required to be adapted every now-and-then. The set of activities fall under this umbrella is termed as maintenance. Regular maintenance of the software deserves deputation of skilled (group of) people who may be from amongst the developers or from amongst the users. To avoid delays and complications, procedural details such as time-lines, payment, mode of payment etc. must be documented properly in this phase. Following factors have the potential to affect the maintenance phase of SDLC [5].

Maintainability, modifiability, Flexibility, and portability are the factors which could affect the maintenance phase of SDLC.

Maintainability: Obviously software cannot be maintained if it is not maintainable. A software may not be maintainable for a number of reasons e.g. an extreme case when the source code is not available. Even if the code is available, maintainability may not be possible. Maintainability of the software gets compromised if it has not been coded properly. Spaghetti type coding has all the potential to ruin maintainability. Undue uses of GOTO type statements are equally harmful.

Modifiability: Modification is as essentially required as maintainability to keep the software workable over a long duration of time. The ability of the software to get modified is termed as modifiability. Software may not be modifiable on the same grounds as noted under maintainability. To some extent the two factors are overlapping. The discriminating aspects are accommodation of the newer requirements whereas in maintainability existing system is required to be adapted to coop with the changing environment [6].

Flexibility: Software has to be flexible enough to get fully used in different environments without needed to be changed. This ability of the software widens its acceptability region and improves its functionality. The flexibility can be induced by giving designing of the software its due attention. Presence of the factor in the software makes it less prone to maintenance and modification.

Portability: Portability is the strategy of writing software to run on one operating system or hardware configuration while being conscious of how it might be refined with minimum effort to run on other operating systems and hardware platforms as well. This is how the software becomes transferable. The acceptability, flexibility, modifiability, and maintainability all goes better if the software is portable.

The four factors namely flexibility, maintainability, modifiability, and portability are directly related to the ability of maintenance of the software. Compromise on any of these factors may affect maintenance. Of course, if software loses maintenance, it cannot be maintained. Similarly if it does not stay flexible no maintenance is possible. Absence of modifiability makes its maintenance impossible. Lack of portability snatches the maintenance.

5. Why certain phases of SDLC there against certain factors?

Now we have to have a look at the Table 2 from a different aspect i.e. we are going to answer the second question we posed at the beginning of this section. The question was "Why certain phases of SDLC there against certain factors?"

The factors are discussed with respect to frequency of their occurrence against various phases of SDLC.

Functionality: This factor has been found listed against System Analysis and Design, Integration/ Testing, Implementation, and Installation and Acceptance. Functionality is the capability of the software product to adhere to standards, conventions, or regulations in laws and similar prescriptions relating to functionality. A closer look at the definition of the factor reveals that functionality has to be care about at System Analysis and Design phase of SDLC without which the phase may not be a success. Integration and/or testing will be a failure if the functionality of the software has been compromised. Implementation of less functional software can be easily ruled out.

Correctness: Relevance of Integration/Testing, Implementation, and Installation and Acceptance phases of SDLC has been reported in the table. No software will be successful software if it is not integrated and/or tested correctly. Incorrectly installed system cannot be accepted. Correct implementation is also essentially required.

Efficiency: System Analysis and Design, Integration/Testing, and Implementation are the three phases of SDLC which were found having relevance to the factor. As it is obvious that without giving proper attention to the System Analysis and Design the efficiency of the final product cannot be met up. As far as Integration and/or Testing are concerned, they heavily rely upon the design of software. Good design makes integration as well as testing of the software efficient because activities' efficiency are ultimately based upon the inherent structure defined at design level. Efficient implementation of the product is guaranteed for efficiently designed, integrated, and tested software [5], [6].

Reliability: Good design and sound testing improve dependence upon the product. System Analysis and Design, and Integration/Testing phases of SDLC, thus, found their way to get listed against the factor.

Contexualness and Descriptiveness: The two factors fall in the Preliminary Investigation phase of SDLC. These factors demands that the requirements are needed to be given in a sufficiently descriptive form so that the software can be managed effectively. For the obvious reason the first phase of SDLC is the most relevant.

Flexibility, Maintainability, Modifiability, and Portability: The four factors listed here are all related to the ability of the software to get maintained over its entire service life. Each factor somehow contributes towards maintenance so it is listed against the Maintenance phase of SDLC.

Human Engineering, Understandability, and Usability: Two of the three listed factors namely Understandability and Usability are related to analysis of the system whereas Human Engineering factor is about design. This makes these factor fall in the System Analysis and Design phase of SDLC.

Integrity: The integrity, hierarchy of access control, has a direct relation to the Implementation phase of SDLC.

Internal: Successful deployment and then running of software is not possible unless requirements have been sought and understood at initial stage. This perspective makes it directly relevant to the Requirement Specification phase of SDLC.

Testability: The obvious direct relation between testability and testing do not deserve any explanation on why this factor is related to Integration/Testing phase of SDLC [6], [5].

6. Conclusion

The quality of software is based on its development. If the phases of System Developing Life Cycle and Factors of Quality Models relate with each other, this can cause a good understanding of the software and it can be easy for the developer to cross check parallel both the developing and quality in his mind. By this the constraint of time can be reduced and quality can improve.

7. REFERENCES

[1] Boehm, B. W.; Brown, J. R.; Kaspar, H.; Lipow, M.; McLeod, G.; and Merritt, M., "Characteristics of Software Quality," *North Holland Publishing, Amsterdam, The Netherlands, 1978, vol., no., pp*.

[2] Deissenboeck, F.; Juergens, E.; Lochmann, K.; and Wagner, S. 2009**.** Software quality models: purposes, usage scenarios and requirements. In *Proceedings of the 7th ICSE conference on Software quality* (Munich, Germany, 2009).

[3] Dromey, R. G., "A Model for Software Product Quality," *IEEE Transactions on Software Engineering*, 1995, vol. 21, no., pp 146-162.

[4] Kitchenham, B., Pfleeger, S. L. "Software Quality: the Elusive Target,"1996. *IEEE Software*, vol. 13**,** no. **,** pp. 12-21.

[5] Pressman, R. S., Software Engineering: A Practitioner's Approach, New York: McGraw-Hill, ISBN: 0071267824 (April 2009).

[6] Al-Qutaish, R. E., "Quality Models in Software Engineering Literature: An Analytical and Comparative Study," *Journal of American Science*, 2010. Vol. 6, no. 3, pp. 166-175.

[7] ISO. ISO/IEC 9126-1: Software Engineering - Product Quality - Part 1*: Quality Model. International Organization for Standardization, Geneva, Switzerland*, 2001.

[8] Voas, J. (2003). *Assuring Software Quality Assurance*. IEEE Software, 20(3), 48-49.

[9] McCall, J. A., Richards, P. K., Walters, G. F. "Factors in Software Quality, Volumes I, II, and III," 1977. *US Rome Air Development Center Reports, US Department of Commerce, USA.*