Relationship between the System Development Life Cycle and Software Quality Criterions for Achieving a Quality Software Product

Basit Habib
Bahauddin Zakariya University
Multan Pakistan

ABSTRACT
To develop a quality software "System development life cycle" is the best technique to be adopted. Software has the ability to map itself on a Quality Model so that its working can be seen on a set of factors along with their criterion. By this correlation better results can be gained from the working of software on every environment. To achieve a quality product it is necessary for the developer to understand how it can be achieved. Theoretical study is the main phase of understanding the performance of software. For achieving a quality product, in this paper the collaboration of SDLC with Criteria of Quality models will be seen for the better working of a software system.

Categories and Subject Descriptors
- Why certain factors are there against SDLC?
- Why certain phases of SDLC there against certain of Quality Models ?
- Combination of the Software Quality Factors corresponding SDLC

General Terms
Reliability, Experimentation, Performance, Design

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

1. INTRODUCTION
1.1 Software Quality Model and the System Development Life Cycle
A software quality model is a set of Factors and Criterion against those factors. The idea of a software quality model is to show such attributes which can make a software work properly. A Quality model is based on set of Factors and different criterions. System Development Life Cycle is basically a technique which is used to develop any kind of a system. 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. 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:

- Code writing.
- Removal of Errors.

Preliminary investigation, Requirement Specification, System analysis and design, are used for the developing or in other words used for code writing of any project, and Installation and acceptance along with maintenance is used for the fix and removal of errors in any project.

1.2 Defining Factor and Criterion
In a quality model the set of factors are similar to bones in a quality model which produce the structure from the beginning to the ending of a Project. And they show the step wise process of a project or work to be done. If the steps are properly integrated into the structure of the skeleton, it becomes more and more strong. Whereas Criterion is the sub sets of a factor as they are the joints of the bones (factors) which show how the steps can show flexibility for the working of the Product to be developed.

2. SOFTWARE QUALITY
Exact definition of software quality is hard to be given because it is difficult to have a consensus arrived between people from different areas of application. Few claim that it is "[the] degree to which a set of inherent characteristics fulfils requirements" (ISO/IEC 1999b) for others it may merely be equivalent to "customer value" (Highsmith, 2002), or even "defect levels" (Highsmith, 2002). This disagreement is simply because people fail to recognize that they are considering different perspectives of quality.

Following perspectives of quality have been reported in the literature namely:
The transcendental perspective which covers the metaphysical aspect of quality. That is, it is "something toward which we strive as an ideal, but may never implement completely." (Kitchenham & Pfleeger, 1996);
The user perspective discusses the appropriateness of the product for a given context of use.

The manufacturing perspective is the degree of conformability against the requirements. It is in line with the standards such as ISO 9001
The product perspective. In this perspective quality can be related to the inherent characteristics of the product. Quality of the end product can this way be achieved by measuring some attributes of different components composing the product.
The value-based perspective. This perspective suggests that various stakeholders may give different weight to the different perspectives of quality on the basis of importance they may be giving to the perspectives.

Overwhelming stress on ISO and IEEE standards give impression that manufacturing perspective is gaining its unduly higher share. The favourable argument in this regard is

that process improvement will lead to improved product quality. The counter argument by Dromey (1996) is *"that the emphasis on process usually comes at the expense of constructing, refining, and using adequate product quality models". This opinion was* seconded by Kitchenham and Pfleeger (1996) who said that conformance to process standards does not guarantee good products but uniformity of the products. Highsmith (2002) have shown that high quality is attainable without stressing too much manufacturing perspective.

The maturity level of organization and quality of the resulting product has been found correlated by Haley (1996). This may be because error/defect density usually gets lowered as maturity improves. It also lowers the time to complete parts of the SDLC and also improves estimation capability. It means that a mature process has all the guts to improve the quality of the software product.

It can be said that one should not rely upon one perspective only rather quality should be gauged by incorporating a suitable blend of all the above-mentioned perspectives. The relative weight of a particular perspective may range from 0 to 1 so as the sum of the weights may not exceed 1. That is, if P_i, where $i = 1,2, \dots, 5$, represents a particular perspective then the blend could be defined as

$$\sum_{i=1}^{5} W_i P_i = W_1 P_1 + W_2 P_2 + W_3 P_3 + W_4 P_4 + W_5 P_5$$

Under the condition that

$\sum_{i=1}^{5} W_i \leq 1$and

$$0 \leq W_i \leq 1$$

This way W_i may be subjectively allocated to meet a particular requirement.

2.1 Why Quality?

For most of today's problem we declare *the computer* the culprit. We do not bother to check if the fault is there in the hardware or in the software which is working at the back. Rather we do not want to distinguish software and hardware we just blame the computer. It is basically the software which causes such problems in our daily lives. This makes increase in public concern about the pervasiveness of software, not only in public services but also in consumer products like automobiles, ATMs, telephones etc. Consequently, we need to worry about quality of all our software products more than ever before.

2.2 Ways of Producing a Software Product

There are several ways to develop a software product. These procedures are collectively termed as Software Development Life Cycle (SDLC) or Software Development Process. Technically individual procedures are called models. Each model describes approaches to a variety of tasks during the process.

It has long been desired that repeatable, predictable processes may be found that improve productivity and quality of the ultimate software product. Systemization and formalization of every minute level is attempted by a section of developers whereas others prefer project management techniques for the purpose. The later approach guarantees meeting time and budget deadlines.

Common ways of developing software products include but limited to the following.

2.2.1 Waterfall model

In this model following steps are required to be followed in succession:

- Requirements analysis
- Software design
- Implementation and Integration
- Testing (or Validation)
- Deployment
- Maintenance

Although after proceeding to the next step one looses the right to review an earlier step but while you are on a particular step you can keep reviewing it as long you do not decide to move to the next step.

2.2.2 Spiral model

If risk management is required then spiral model should be used as risk management at regular stages in development cycle is its key characteristic. This model is the blend of waterfall model and rapid prototyping methodologies and emphasizes on deliberate iterative risk analysis.

The process can be visualized as a process passing through a number of iterations to iterate: formulate plans to identify software targets, selected to implement the program and to clarify the project development restrictions; Risk analysis to consider how to identify and eliminate risk; the implementation of software development and verification.

2.2.3 Iterative and incremental development

Iterative development starts with the construction of initially smaller but grows to larger portions of a software product. This helps all team members to foresee important issues and pre-empt a possible disaster.

2.2.4 Agile development

Agile software development uses iterative development as a basis but uses feedback, rather than planning, as their primary control mechanism. It produces as it gets evolved on the basis of regular tests and releases of the evolving software.

2.3 Keeping Track of Software Development

Software Development is the set of activities that includes everything from conception of the desired software through manifestation of the software. Thus it may include research, new development, prototyping, modification, ruse, re-engineering, maintenance, or any other activities that result in a software product.

2.4 Why to Keep Track of Software Development

The quality of the ultimate software product depends on the process of development as well as on the level of care exercised while the product was being developed. That is, the quality is not anything which can be dressed-up after the product has produced. The aspect of quality has to be taken care of at each stage of development. Moreover, if a software team stresses quality in all software engineering activities, it reduces the amount of rework that it must do. Ultimately it not only cuts the cost but it cuts the time to deliver. In short, the quality depends on the quality control of the process.

2.5 What Makes Software, "Quality Software"?

From the end-users' aspect of quality, good software should have following qualities:

The end-user should not be required to know the details of the software or needs to have access to the source code. He must not need to establish any contact with the developers.
The software must be substantially and correctly documented from the prospective user's viewpoint without presuming special knowledge available to the developers.
The software should be substantially error free. It should not work only when used by the developers.
The software should have been rigorously tested and must be able to perform under a wide variety of situations.
Sufficient error handling and input and data validation must be an integral part of software.

2.6 Testing a Software Product

The developers are bound to made inadvertent errors while developing a software product. These errors are required to be uncovered and then adjusted before the product is delivered. Several questions deserve their due attention during this stage. For example, how the test should be conducted? Do we need to develop a formal plan for the test? If the entire program as a whole should be tested or tests be performed only on a smaller parts? Do we need to rerun test after every component is added? Do we need to involve the customer?

2.6.1 Testing Strategies

2.6.1.1 Unit Testing or Component Testing

It refers to tests that verify the functionality of a specific section of code (in conventional design at the function level and in object oriented design at the class level). These type of tests use white-box style as those are developed by the developers during development phase to make sure that the code is working correctly. It is likely that the unit testing alone may not provide exhaustive results but it does assure that the building blocks satisfactorily in stand-alone position.

2.6.1.2 Integration Testing

When stand-alone components are integrated it is likely that they may repel each other or they lose data interface. It is good to test the integrated components as an entity. Integration testing does the job. That is Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. This requires the knowledge how components have been integrated? Incremental (bottom-up or top-down) or plunged? Regression testing and smoke testing are typical examples of integration testing.

2.6.1.3 Testing Methods

The systematic measurements of a product are the ways how it is tested to achieve the assurance of quality. Various methods have been introduced for the purpose

White-box testing

It is a method that tests internal structures or workings of a component or an aggregate of components instead of its functionality.

White-box test design techniques include:

- Control flow testing
- Data flow testing
- Branch testing
- Path testing

Black-box testing

It is a method that tests the functionality of a component or aggregate of components instead of its internal structures or workings. Access to application's code, knowledge of internal structure, or understanding of programming knowledge is not required. The only requirement is the knowledge of what the software is supposed to do, but not how to do.

Typical black-box test design techniques include:

- Decision table testing
- All-pairs testing
- State transition tables
- Equivalence partitioning
- Boundary value analysis
- Use case testing

Grey-box testing

It is blend of two opponent test methods namely white-box and black-box testing. The aim of this testing method is to look for the possible defects due to improper structure or improper usage of applications. It is also termed as *translucent testing*

Exploratory testing

Cem Kaner defines it as "*It is a style of software testing that emphasizes the personal freedom and responsibility of the individual tester to continually optimize the quality of his work by treating test-related learning, test design, test execution, and test result interpretation as mutually supportive activities that run in parallel throughout the project.*"

Thus, it can be regarded as an evolutionary test procedure which gets improved at every stage of evolution and attains ultimate target of quality.

2.7 Metrics and Their Role

2.7.1 Metrics

The fundamental question is what are *Quality Metrics*? As *metric* is a measure and *Quality* is something desired by the customer thus a *Quality Metric*, therefore, is a measure of quality as defined by the customer. In this sense metrics are usually considered quantitative measures but qualitative metrics do exist. On one hand accuracy of inventory can be reported as accurate as 90% and on the other hand performance of a company can be gauged as *above average*. So creating metrics for quality can either be qualitative or quantitative. As the main idea is to be able to obtain excellent performance with regards to the quality of the products or the services so quality metrics provides basis to achieve the set quality of the software products.

2.7.2 Common Quality Metrics

2.7.2.1 Correctness

Correctness is the degree to which the software performs the required functions accurately. It may be measured as Defects per Kilo (Thousands) of Lines Of Code (KLOC).

2.7.2.1 Maintainability

Maintainability is the measure of level of ease with which a program can be corrected if an error occurs. Mean Time To Change (MTTC) is used to measure it.

2.7.2.3 Integrity

It measures the system's ability to manage its security. Its measure depends upon probability that an attack of certain type will happen over a period of time (Threat) and probability that an attack of certain type will be removed over a period of time (Security).

2.7.2.4 Usability

The usability of software application is assessed in term of skills required to learn the system, time required to become moderately efficient in the system, the net increase in productivity by use of the new system, and subjective assessment.

3. Software Quality Models and their Certain Criterion

3.1 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.

3.2 Explanation of Quality Models:

3.2.1 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].

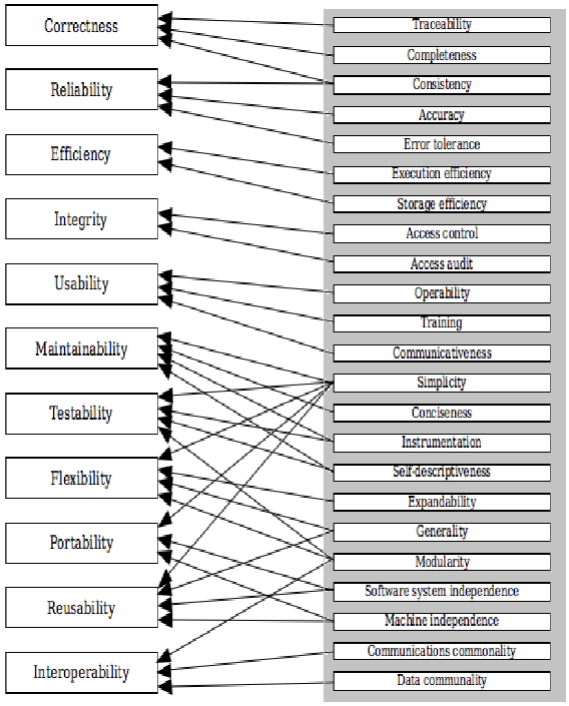

Figure 1. McCall's Quality Model Adapted from McCall (1977) and Pfleeger(2003)

3.2.2 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]

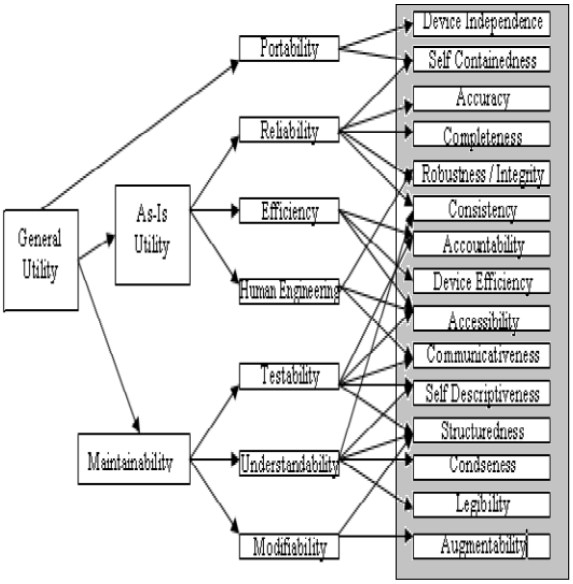

Figure 2. Boehm's Quality Model Adapted from Boehm(1978) and Pfleeger(2003)

3.2.3 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.

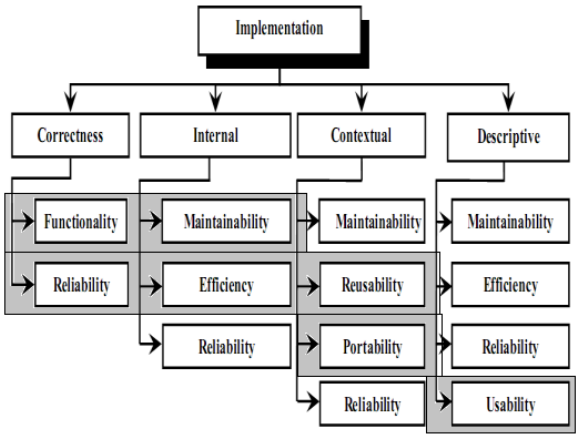

Figure 3. Dromy's Quality Model Adapted from Dromey(1996) and Pfleeger(2003)

4. CRITERIA OF SOFTWARE QUALITY MODEL

Table 1. Mapping of Criterion on Phases of SDLC.

Phases of SDLC / Quality Criterion	Preliminary Investigation	Requirement Specification	System Analysis and Design	Integration\ Testing	Implementation	Installation and Acceptance	Maintenance
Consistency			v				
Modularity							
Self Descriptiveness	v						
Simplicity			v				
Accuracy	v				v	v	
Communicativeness		v					
Compliance	v			v			
Structurdeness			v				
Access Audit					v	v	
Access Control					v	v	
Accessibility						v	
Adoptability				v		v	
Augmentability	v			v			
Co-Existence	v	v	v	v	v	v	v
Commonality	v	v	v	v	v	v	v
Completeness	v	v	v	v	v	v	v
Error Tolerance			v	v		v	
Execution Efficiency					v	v	
Expandability							v
Generality	v		v	v			
HW and SW independence	v	v				v	
Insatiability						v	
Instrumentation		v					
Integrity						v	
Inter Operability	v				v		
Legibility	v	v					
Operability						v	
Replace ability							
Security						v	
Storage Efficiency		v				v	
Suitability			v	v		v	
Tractability	v			v		v	
Training						v	v

4.1 Consistency

Consistency is one of the basic criteria which show how to keep on the right path as declare during system development. By seeing the phases of SDLC and mapping consistency in them "System Analysis and Design" is the phase in which if consistency is kept then the issue of time achievement can be gained respectively

4.2 Modularity

This criterion shows that in how many modules a problem statement for designing a system has to be divided. By using modularity in "Preliminary Investigation" in the beginning of the problem while it can be judged that in how many parts (modules) the problem has to be divided. Modularity is also a major part of "System Analysis and Design" as it plays the role of building blocks in the structure of a system .Also in the phase of "Integration\Testing, modularity can be used as for the correct synchronization of the system.

4.4 Self Descriptiveness

This criterion shows the elaboration of the problem statement as well as the road map defines to design the system. The phase "Preliminary Investigation" fulfils the first part of self descriptiveness where as "Requirement Specification" fulfill the second part of Self Descriptiveness.

4.4 Simplicity

Simplicity is such a criteria which shows the easiness and user friendliness in a system. Simplicity exists in "System Analysis and Design" because in this phase of SDLC it depend on the

developer if he follows the two above mentioned rules of simplicity for a better system.

4.5 Accuracy

This is one of the most important criteria of a software quality model. As seen if this criterion is followed from phase to phase of SDLC then the system has minimum chances of failure. Accuracy is a very 'versatile' criterion due to its occurrence in every phase of SDLC. As regarding this criterion the function of 'Recursive Call' cannot be considered again and again so as the name describes in the criteria accuracy, accurate judgment should be held.

4.6 Communicativeness

This criterion shows how the different modules of a system will communicate with each other by using the right specified kind of tool. If we see the above mentioned statement it is easy to judge that "requirement specification" and "Installation and Acceptance" are the two main phases which poses a strong communicative bond between them.

4.7 Compliance

This criterion shows the complete hypothetical road map and desired hypothesis regarding the output of a system. The phases " Preliminary Investigation" and " Integration\Testing" show how to design the system in a stepwise process and testing tells the prone and cones regarding the output of that system.

4.8 Structurdness

This criterion depends on the size of the works neumacleature of every single module which is going to be a part depending on the size of that system. The phase "System Analysis and Design" shows how to develop every single structure of a specific module and "Integration\ Testing" shows how every structure co\relate with each other.

4.9 Access Audit \ Access Control

These two criterions show that where and how to give implementation access to the system and who will have the excessive control of that system. The phases of "implementation and installation" and "acceptance" and "maintenance" are those phases which use these both criterions.

4.10 Accessibility

This criterion tell that how the end user will have a fully functional access to the system after its implementation .Because after implementation the end user can have access to the system therefore the phase "Installation and acceptance" will show how the criteria of accessibility works after a specific process for the end user.

4.11 Adoptability

This criterion is again one of the most important of SDLC. This criteria shows how either a module, a testing technique and even the end user will show the adeptness towards the system. By seeing adoptability in SDLC "Integration\Testing" and "Installation and Acceptance" carry the importance of module and end user adoptability.

4.12 Augment ability

This criterion basically shows that the work done on specific levels is either appropriate or not. For instance was "preliminary Investigation" done correctly in understanding the problem statement? Was "Integration\Testing" done correctly for every single module designed to develop the system.

4.14 Co- existence \ Commonality

This criterion is one of the time saving methods if which adopted will gain all those familiar activities, techniques and even modules which exist similar in SDLC. We can say that if we have four modules in a system and three module out of four use a specific similar kind of testing technique or designing technique regarding its algorithm or programming language then the criteria of co-existence \ commonality is seen.

4.14 Completeness

This criterion is seen on every single phase of SDLC, but is always a question mark due to the disordered behavior of phases of SDLC. Discussing disorder we can say that the term of recursive call and the unpredictable mind setup of the end user, this criterion is fulfilled by the developer but always it is a question mark for the end user.

4.15 Error Tolerance

By seeing the criterion of completeness and the unpredictable mind setup of the end user this criteria of error tolerance is implemented in integration\ testing, implementation, installation and acceptance are the phases where error tolerance is accepted by the end user.

4.16 Execution Efficiency

This criterion shows how efficiently a project is executed from the first to the last phase of SDLC. Because quick output has always been the running demand of the end user therefore due to this criteria the end user bares his lose in error tolerance.

4.17 Expandability

This criterion tells that how much a system is expandable with the demand of time. This criterion exists in "maintance" and has an indirect relationship with" System Analysis and Design".

4.18 Instrumentation

This criterion is directly related to "requirement specification". As in this phases the developer describes the required tools for designing and acceptance of the system.

4.19 Integrity

This criterion shows that all the by parts distribution of the problem statement in different modules after development is joined in a proper manner. By observing the above lines it is easy to access that integrity tell us about the proper joining of modules for the correct working of the system.

5. CONCLUSION

Mainly it is a common technique that mostly Phases of SDLC are mapped on the Factors of a Quality Model, but in this paper it is for the first time that The different Criterion of different Quality models have been mapped on the phases of SDLC and by doing such kind of work it is seen that the quality of a product can be achieved in a new trend. But it is earlier to say whether Mapping of Factors on SDLC is more sufficient or Mapping of Quality Criterion.

6. REFERENCES

[1] Software Engineering- Product Quality, Part 3, 4, Version 1. 0 (ISO/IEC TR 9126), url: http://www. iso. org/iso/home/store/catalogue_tc/, visited: 7th July, 2012.

[2] 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.

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

[4] Wirth, N. "A Brief History of Software Engineering," IEEE Annals of the History of Computing, 2008. Vol. 30 no. 3, pp. 32-39.

[5] 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.

[6] 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).

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

[8] History of Software Engineering. In Wikipedia. Retrieved June 27 2012, from http://en. wikipedia.org/wiki/History_of_software_engineering.

[9] Software Engineering- Product Quality, Part 3, 4, Version 1. 0 (ISO/IEC TR 9126), url: http://www.iso. org/iso/home/store/catalogue_tc/, visited: 7th July, 2012.

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

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

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

[13] Compilation of Software Quality Factors and Criteria along with their Description for a Quality Product, B Habib, International Journal of Computer Applications 84 (3), 22-27

[14] Relationship between Factors of Quality Models and the System Development Life Cycle B Habib, Int. Journal of Computer Applications (IJCA) 81 (Software Engg.), 39-44