Quality Models in Software Engineering Literature: An Analytical and Comparative Study

Rafa E. Al-Qutaish, PhD

Al Ain University of Science and Technology – Abu Dhabi Campus, PO Box: 112612, Abu Dhabi, UAE.
rafa@ieee.org

Abstract: The quality of the software is critical and essential in different types of organizations. In some types of software, poor quality of the software product in sensitive systems (such as: real-time systems, control systems, etc.) may lead to loss of human life, permanent injury, mission failure, or financial loss. In software engineering literature, there are a number of quality models in which they contain a number of quality characteristics (or factors, as called in some models). These quality characteristics could be used to reflect the quality of the software product from the view of that characteristic. Selecting which one of the quality models to use is a real challenge. In this paper, we will discuss the contents of the following quality models: McCall's quality model, Boehm's quality model, Dromey's quality model, FURPS quality model and ISO 9126 quality model. In addition, we will focus on a comparison between these quality models, and find the key differences between them. [Journal of American Science 2010; 6(3):166-175]. (ISSN: 1545-1003).

Keywords: Software Quality; Quality Models; Quality Engineering; ISO 9126; McCall's Quality Model; Boehm's Quality Model; Dromey's Quality Model; FURPS Quality Model

1. Introduction

Software is critical in providing a competitive edge to many organizations, and is progressively becoming a key component of business systems, products and services. The quality of software products is now considered to be an essential element in business success [Veenendaal and McMullan, 1997]. Furthermore, the quality of software product is very important and essential since for example in some sensitive systems – such as, real-time systems, control systems, etc. – the poor quality may lead to financial loss, mission failure, permanent injury or even loss of human life.

There are several definitions for "software Quality" term, for examples, it is defined by the IEEE [1990] as the degree to which a system, component or process meets specified requirements and customer (user) needs (expectations). Pressman [2004] defines it as "conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software." The ISO, by contrast, defines "quality" in ISO 14598-1 [ISO, 1999] as "the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs," and Petrasch [1999] defines it as "the existence of characteristics of a product which can be assigned to requirements."

There are a number of quality models in software engineering literature, each one of these quality models consists of a number of quality characteristics (or factors, as called in some models). These quality characteristics could be used to reflect the quality of the software product from the view of that characteristic. Selecting which one of the quality models to use is a real challenge. In this paper, we will discuss the contents of the following quality models:

1. McCall's Quality Model.
2. Boehm's Quality Model.
3. Dromey's Quality Model.
4. FURPS Quality Model.
5. ISO 9126 Quality Model.

In addition, we will focus on a comparison between these quality models, and find the key differences between them.

The rest of this paper is structured as follows: Section 2 presents an overview of the five common quality models used in software engineering. Section 3 contains a detailed analysis and comparison between the five quality models. Finally, Section 4 concludes the paper with some comments.

2. An Overview of the Software Quality Models

2.1 McCall's Quality Model

McCall's Quality Model (also known as the General Electrics Model of 1977) is one of the most known quality models in the software engineering literature. It has been presented by Jim McCall et al. [1977]. This model originates from the US military and is primarily aimed towards the system developers and the system development process [McCall et al, 1977]. Using this model, McCall attempts to bridge the gap between users and developers by focusing on a number of software quality factors that reflect both the users' views and the developers' priorities [McCall et al, 1977].

The structure of the McCall's quality model consists of three major perspectives (types of quality characteristics) for defining and identifying the quality of a software product, and each of these major perspectives consists of a number of quality factors. Each of these quality factors has a set of quality criteria, and each quality criteria could be reflected by one or more metrics, see Figure 1 for the details of the McCall's quality model structure. The contents of the three major perspectives are the following:

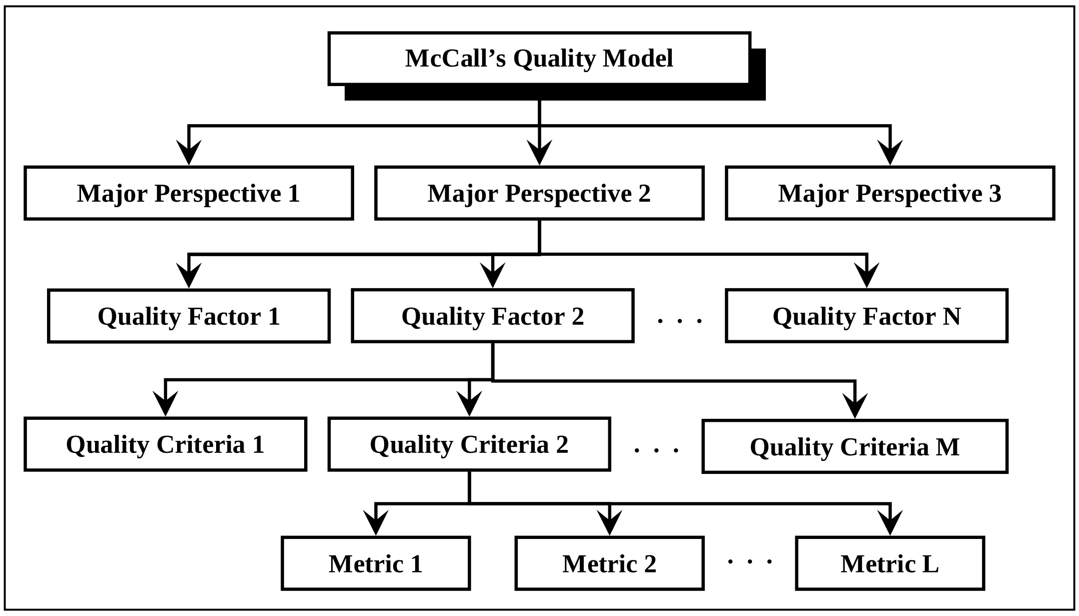

Figure 1. The structure of McCall's quality model

1. Product Revision: it is about the ability of the product to undergo changes, and it includes:
   a. Maintainability: the effort required to locate and fix a fault in the program within its operating environment.
   b. Flexibility: the ease of making changes required by changes in the operating environment.
   c. Testability: the ease of testing the program, to ensure that it is error-free and meets its specification.
2. Product Operations: it is about the characteristics of the product operation. The quality of the product operations depends on:
   a. Correctness: the extent to which a program fulfils its specification.
   b. Reliability: the system ability not to fail.
   c. Efficiency: it further categorized into execution efficiency and storage efficiency and generally meaning the use of resources, e.g. processor time, storage.
   d. Integrity: the protection of the program from unauthorized access.
   e. Usability: the ease of the use of the software.
3. Product Transition: it is about the adaptability of the product to new environments. It is all about:
   a. Portability: the effort required to transfer a program from one environment to another.
   b. Reusability: the ease of reusing software in a different context.
   c. Interoperability: the effort required to couple the system to another system.

In more details, McCall's Quality Model consists of 11 quality factors to describe the external view of the software (from the users' view), 23 quality criteria to describe the internal view of the software (from the developer's view) and a set of Metrics which are defined and used to provide a scale and method for measurement. Table 1 presents two of the three major perspectives and their corresponding quality factors and quality criteria.

The main objective of the McCall's Quality Model is that the quality factors structure should provide a complete software quality picture [Kitchenham, 1996]. The actual quality metric is computed by answering "yes" and "no" questions. However, if answering equally amount of "yes" and "no" on the questions measuring a quality criteria, then you will achieve 50% on that quality criteria.

Table 1. The contents of McCall's quality model - product revision and product operations

Major Perspectives	Quality Factors	Quality Criteria
Product revision	Maintainability	Simplicity
		Conciseness
		Self-descriptiveness
		Modularity
	Flexibility	Self-descriptiveness
		Expandability
		Generality
	Testability	Simplicity
		Instrumentation
		Self-descriptiveness
		Modularity
Product operations	Correctness	Traceability
		Completeness
		Consistency
	Efficiency	Execution efficiency
		Storage efficiency
	Reliability	Consistency
		Accuracy
		Error tolerance
	Integrity	Access control
		Access audit
	Usability	Operability
		Training
		Communicativeness

2.2 Boehm's Quality Model

Boehm [1976, 1978] introduced his quality model to automatically and quantitatively evaluate the quality of software. This model attempts to qualitatively define the quality of software by a predefined set of attributes and metrics. It consists of high-level characteristics, intermediate-level characteristics and lowest-level (primitive) characteristics which contribute to the overall quality level (see Figure 2).

In this model, the high-level characteristics represent basic high-level requirements of actual use to which evaluation of software quality could be put. In its high-level, there are three characteristics, that is [Boehm et al, 1976, Boehm et al, 1978]:

1. As-is utility: to address how well, easily, reliably and efficiently can I use the software product as-is?
2. Maintainability: to address how easy is it to understand, modify and retest the software product?
3. Portability: to address if can I still use the software product when the environment has been changed?

Table 2 shows the contents of the Boehm's quality model in the three levels, high-level, intermediate-level and lowest-level characteristics. In addition, it is noted that there is a number of the lowest-level characteristics which can be related to more than one intermediate-level characteristics, for example, the 'Self Contentedness' primitive characteristic could be related to the 'reliability' and 'portability' primitive characteristics.

In the intermediate level characteristic, there are seven quality characteristics that together represent the qualities expected from a software system [Boehm et al, 1976, Boehm et al, 1978]:

1. Portability: the software can be operated easily and well on computer configurations other than its current one.
2. Reliability: the software can be expected to perform its intended functions satisfactorily.
3. Efficiency: the software fulfills its purpose without waste of resources.
4. Usability: the software is reliable, efficient and human-engineered.
5. Testability: the software facilitates the establishment of verification criteria and supports evaluation of its performance.
6. Understandability: the software purpose is clear to the inspector.
7. Flexibility: the software facilitates the incorporation of changes, once the nature of the desired change has been determined.

The primitive characteristics can be used to provide the foundation for defining quality metrics, this use is one of the most important goals established by Boehm when he constructed his quality model. One or more metrics are supposed to measure a given primitive characteristic. Boehm [1978] defined the 'metric' as "a measure of extent or degree to which a product possesses and exhibits a certain (quality) characteristic."

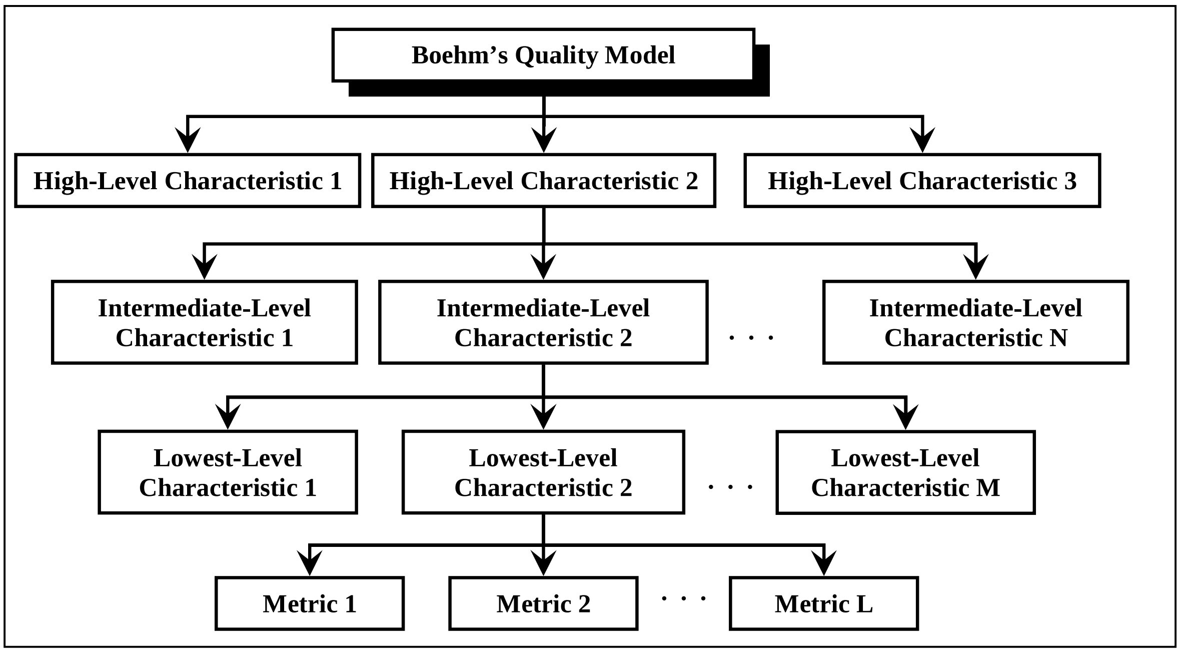
Boehm's Quality Model
High-Level Characteristic 1
High-Level Characteristic 2
High-Level Characteristic 3
Intermediate-Level Characteristic 1
Intermediate-Level Characteristic 2
. . .
Intermediate-Level Characteristic N
Lowest-Level Characteristic 1
Lowest-Level Characteristic 2
. . .
Lowest-Level Characteristic M
Metric 1
Metric 2
. . .
Metric L

Figure 2. The structure of Boehm's quality model

Table 2. The contents of Boehm's quality model

High-Level Characteristics	Intermediate-Level Characteristics	Primitive Characteristics
As-is Utility	Reliability	Self Containedness Accuracy Completeness Robustness/Integrity Consistency
	Efficiency	Accountability Device Efficiency Accessibility
	Human Engineering	Robustness/Integrity Accessibility Communicativeness
Portability		Device Independence Self Containedness
Maintainability	Testability	Accountability Communicativeness Self Descriptiveness Structuredness
	Understandability	Consistency Structuredness Conciseness Legibility
	Modifiability	Structuredness Augmentability
3 High-Level Characteristics	**7 Intermediate-Level Characteristics**	**15 Distinct Primitive Characteristics**

2.3 Dromey's Quality Model

This quality model has been presented by Dromey [1995, 1996]. It is a product based quality model that recognizes that quality evaluation differs for each product and that a more dynamic idea for modeling the process is needed to be wide enough to apply for different systems [Dromey, 195]. Furthermore, Figure 3 shows that it consists of four software product properties and for each property there is a number of quality attributes. In addition, figure 4 shows the contents of the Dromey's quality model.

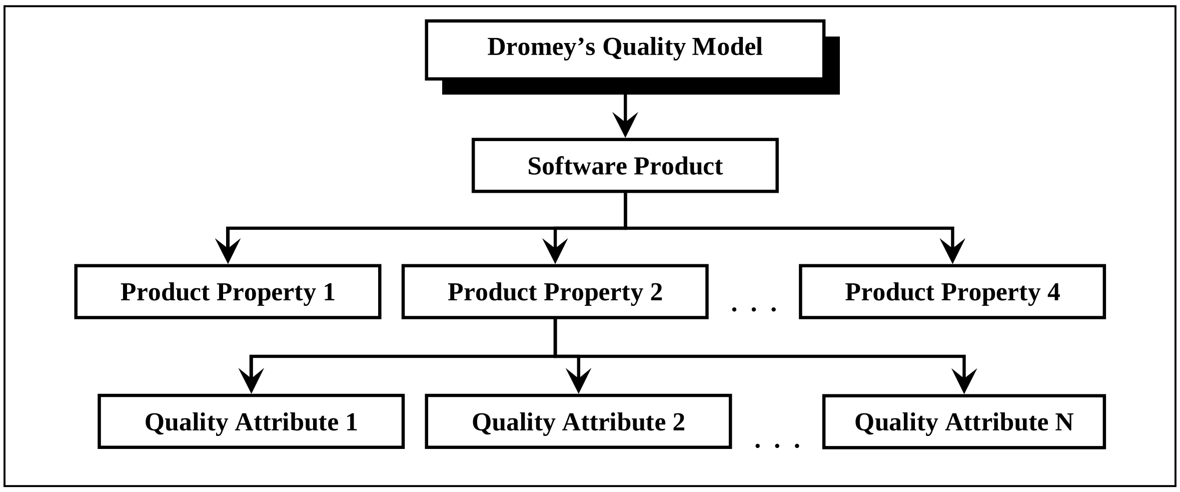

Figure 3. The structure of Dromey's quality model

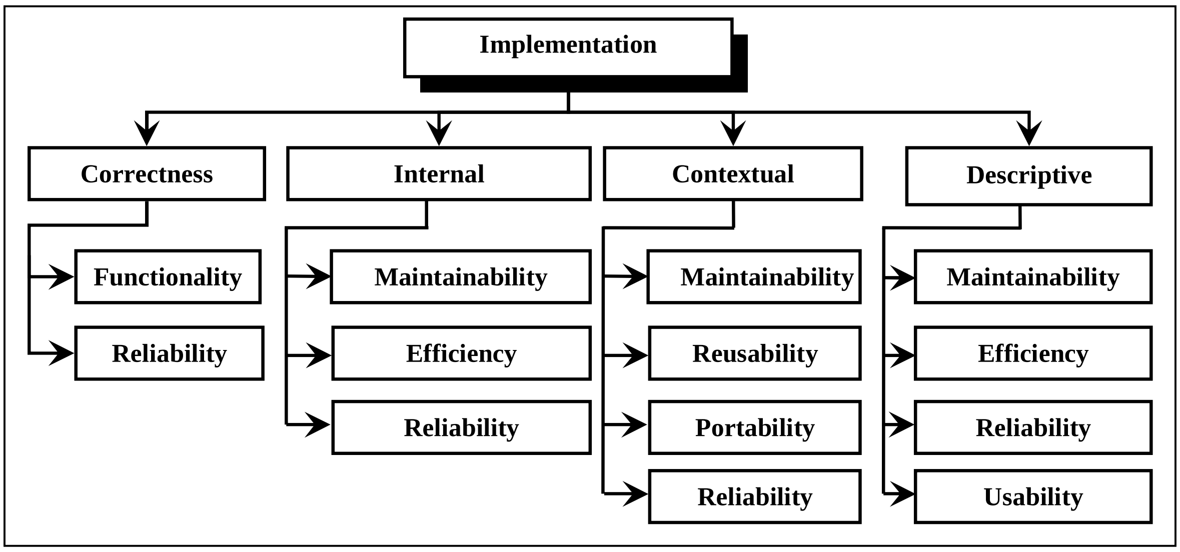

Figure 4. The contents of Dromey's quality model

2.4 FURPS Quality Model

The FURPS model originally presented by Robert Grady[1992], then it has been extended by IBM Rational Software [Jacobson et al, 1999, Kruchten, 2000] into FURPS+, where the '+' indicates such requirements as design constraints, implementation requirements, interface requirements and physical requirements [Jacobson et al, 1999].

In this quality model, the FURPS stands for [Grady, 1992] - as in Figure 5 - the following five characteristics:

1. Functionality: it may include feature sets, capabilities, and security.
2. Usability: it may include human factors, aesthetics, consistency in the user interface, online and context sensitive help, wizards and

agents, user documentation, and training materials.
3. Reliability: it may include frequency and severity of failure, recoverability, predictability, accuracy, and mean time between failures (MTBF).
4. Performance: it imposes conditions on functional requirements such as speed, efficiency, availability, accuracy, throughput, response time, recovery time, and resource usage.
5. Supportability: it may include testability, extensibility, adaptability, maintainability, compatibility, configurability, serviceability, installability, and localizability.

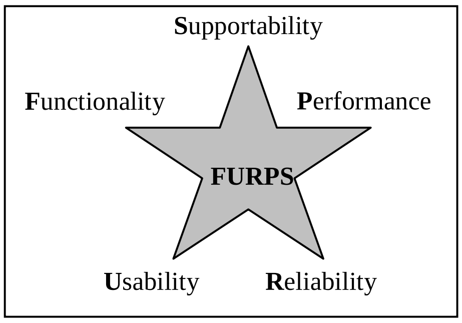

Figure 5. The contents of FURPS quality model

2.5 ISO 9126 Quality Model

In 1991, the ISO published its first international consensus on the terminology for the quality characteristics for software product evaluation; this standard was called as Software Product Evaluation - Quality Characteristics and Guidelines for Their Use (ISO 9126) [ISO, 1991]. From 2001 to 2004, the ISO published an expanded version, containing both the ISO quality models and inventories of proposed measures for these models. The current version of the ISO 9126 series now consists of one International Standard (IS) and three Technical Reports (TRs):
1. ISO IS 9126-1: Quality Model [ISO, 2001].
2. ISO TR 9126-2: External Metrics [ISO, 2003].
3. ISO TR 9126-3: Internal Metrics [ISO, 2003].
4. ISO TR 9126-4: Quality in Use Metrics [ISO, 2004].

The first document of the ISO 9126 series – Quality Model – contains two-parts quality model for software product quality [ISO, 2001]:
1. Internal and external quality model.
2. Quality in use model.

The first part of the two-parts quality model determines six characteristics in which they are subdivided into twenty-seven sub-characteristics for internal and external quality, as in Figure 6 [ISO, 2001]. These sub-characteristics are a result of internal software attributes and are noticeable externally when the software is used as a part of a computer system. The second part of the two-part model indicates four quality in use characteristics, as in Figure 7 [ISO, 2001].

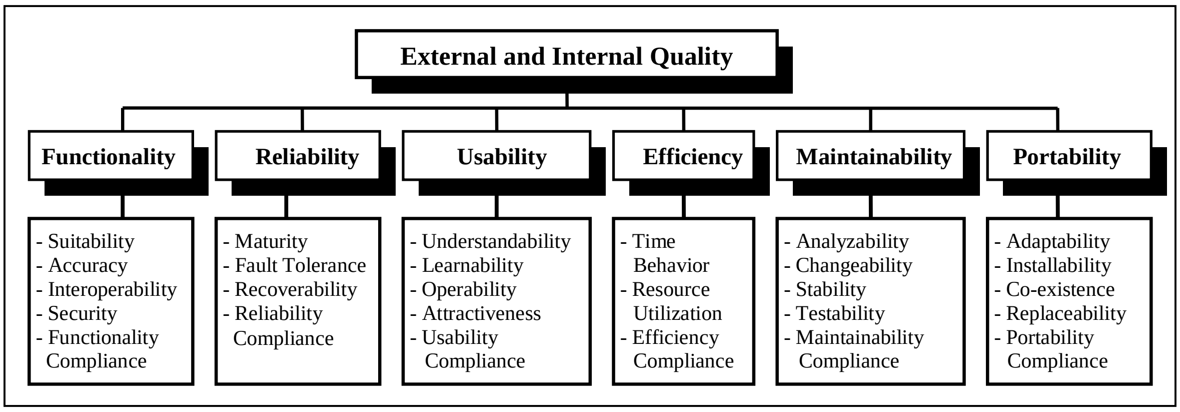

Figure 6. ISO 9126 quality model for external and internal quality (characteristics/sub-characteristics) [ISO, 2001]

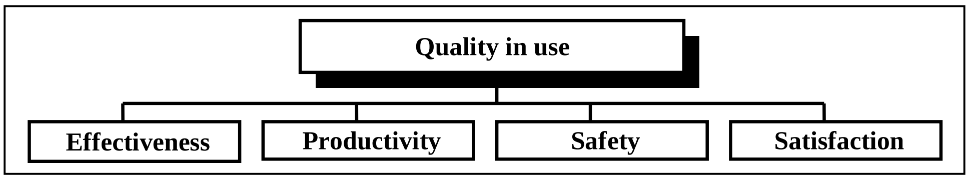

Figure 7. ISO 9126 quality model for quality in use (characteristics) [ISO, 2001]

Figure 8 shows the ISO view of the expected relationships between internal, external, and quality in use attributes. The internal quality attributes influence on the external quality attributes while the external attributes influences on the quality in use attributes. Furthermore, the quality in use depends on the external quality while the external quality depends on the internal quality [ISO, 2001].

For the internal and external software products, each quality characteristics and its corresponding sub-characteristics are defined in ISO 9126-1 [ISO, 2001] as follows:

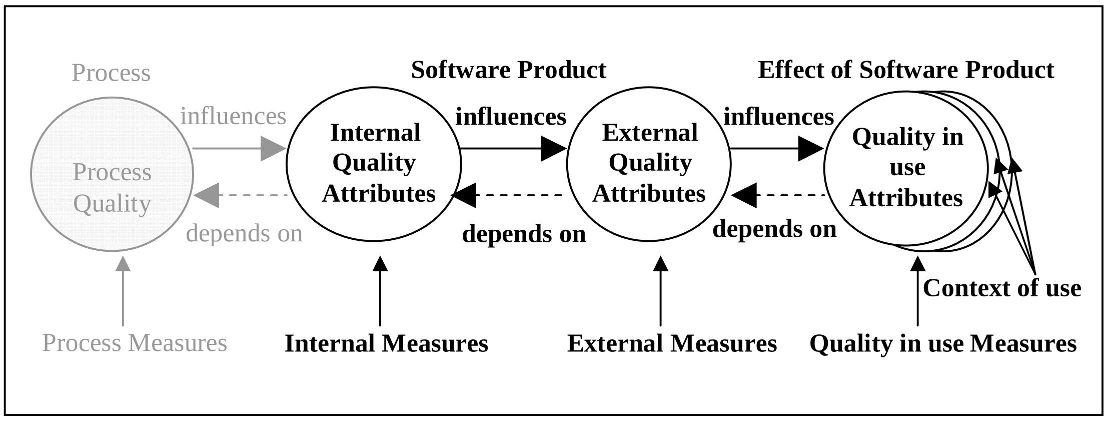

Figure 8. Quality in the lifecycle [ISO, 2001]

1. Functionality: “the capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions”. It contains the following sub-characteristics:
   a. Suitability: “the capability of the software product to provide an appropriate set of functions for specified tasks and user objectives”.
   b. Accuracy: “the capability of the software product to provide the right or agreed results or effects with the needed degree of precision”.
   c. Security: “the capability of the software product to protect information and data so that unauthorised persons or systems cannot read or modify them and authorised persons or systems are not denied access to them”.
   d. Interoperability: “the capability of the software product to interact with one or more specified systems”.
   e. Functionality Compliance: “the capability of the software product to adhere to standards, conventions or regulations in laws and similar prescriptions relating to functionality”.
2. Reliability: “The capability of the software product to maintain a specified level of performance when used under specified conditions”. It includes the following sub-characteristics:
   a. Maturity: “the capability of the software product to avoid failure as a result of faults in the software”.
   b. Fault tolerance: “the capability of the software product to maintain a specified level of performance in cases of software faults or of infringement of its specified interface”.
   c. Recoverability: “the capability of the software product to re-establish a specified level of performance and recover the data directly affected in the case of a failure”.
   d. Reliability Compliance: “the capability of the software product to adhere to standards, conventions or regulations relating to reliability”.
3. Usability: “the capability of the software product to be understood, learned, used, and attractive to the user, when used under specified conditions”. It contains the following sub-characteristics:
   a. Understandability: “the capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use”.
   b. Learnability: “the capability of the software product to enable the user to learn its application”.
   c. Operability: “the capability of the software product to enable the user to operate and control it”.
   d. Attractiveness: “the capability of the software product to be attractive to the user”.
   e. Usability Compliance: “the capability of the software product to adhere to standards,

conventions, style guides or regulations relating to usability".

4. Efficiency: "the capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions". It includes the following sub-characteristics:
   a. Time behaviour: "the capability of the software product to provide appropriate response and processing times and throughput rates when performing its function, under stated conditions".
   b. Resource behaviour: "the capability of the software product to use appropriate amounts and types of resources when the software performs its function under stated conditions".
   c. Efficiency Compliance: "the capability of the software product to adhere to standards or conventions relating to efficiency".
5. Maintainability: "the capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications". It contains the following sub-characteristics:
   a. Analyzability: "the capability of the software product to be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified".
   b. Changeability: "the capability of the software product to enable a specified modification to be implemented".
   c. Stability: "the capability of the software product to avoid unexpected effects from modifications of the software".
   d. Testability: "the capability of the software product to enable modified software to be validated".
   e. Maintainability Compliance: "the capability of the software product to adhere to standards or conventions relating to maintainability".
6. Portability: "the capability of the software product to be transferred from one environment to another". It includes the following sub-characteristics:
   a. Adaptability: "the capability of the software product to be adapted for different specified environments without applying actions or means other than those provided for this purpose for the software considered".
   b. Installability: "the capability of the software product to be installed in a specified environment".
   c. Co-existence: "the capability of the software product to co-exist with other independent software in a common environment sharing common resources".
   d. Replaceability: "the capability of the software product to be used in place of another specified software product for the same purpose in the same environment".
   e. Portability Compliance: "the capability of the software product to adhere to standards or conventions relating to portability".

3. Analysis of the Quality Models

In this section, a comparison between the availability of the characteristics (called factors or attributes in some quality models) within the five quality models will be presented. Table 3 presents this comparison, at the end this table you will find the number of the corresponding characteristics for each quality model.

From the 17 characteristics, only one characteristic is common to all quality models, that is, the 'reliability'. Also, there are only three characteristics (i.e. 'efficiency', 'usability' and 'portability') which are belonging to four quality models. Two characteristic is common only to three quality models, that is, the 'functionality' and 'maintainability' characteristics. Two characteristic belong to two quality models, that is, the 'testability' and 'reusability' characteristics. And, nine characteristics (i.e. 'flexibility', 'correctness', 'integrity' and 'interoperability' in McCall's quality model; 'human engineering', 'understandability' and 'modifiability' in Boehm's quality model; 'performance' and 'supportability' in FURPS quality model) are defined in only one quality model.

Furthermore, it can be noted that the 'testability', 'interoperability' and 'understandability' are used as factors/attributes/characteristics in some quality models. However, in ISO 9126-1, these factors/attributes/characteristics are defined as sub-characteristics. More specifically, the 'testability' is belonging to the 'maintainability' characteristic, the 'understandability' is belonging to the 'usability' characteristic, and the 'interoperability' is belonging to the 'functionality' characteristic.

From our point of view, the ISO 9126-1 quality model is the most useful one since it has been built based on an international consensus and agreement from all the country members of the ISO organization.

Table 3. A comparison between the five quality models

Factors/Attributes/Characteristics	**McCall**	**Boehm**	**Dromey**	**FURPS**	**ISO 9126**
Maintainability	✓		✓		✓
Flexibility	✓				
Testability	✓	✓			
Correctness	✓				
Efficiency	✓	✓	✓		✓
Reliability	✓	✓	✓	✓	✓
Integrity	✓				
Usability	✓		✓	✓	✓
Portability	✓	✓	✓		✓
Reusability	✓		✓		
Interoperability	✓				
Human Engineering		✓			
Understandability		✓			
Modifiability		✓			
Functionality			✓	✓	✓
Performance				✓	
Supportability				✓	
17	**11**	**7**	**7**	**5**	**6**

4. Discussion

There are a number of quality models in software engineering literature, each one of these quality models consists of a number of quality characteristics (or factors, as called in some models). These quality characteristics could be used to reflect the quality of the software product from the view of that characteristic. Selecting which one of the quality models to use is a real challenge. In this paper, we have discussed and compared the following quality models:

1. McCall's Quality Mode.
2. Boehm's Quality Model.
3. Dromey's Quality Model.
4. FURPS Quality Model.
5. ISO 9126 Quality Model.

Based on the discussion of the five quality models and on the comparison between them, the following comments could be written:

1. In McCall's quality model, the quality is subjectively measured based on the judgment on the person(s) answering the questions ('yes' or 'no' questions).
2. Three of the characteristics are used in the ISO 9126-1 quality model as sub-characteristics from other characteristics.
3. The FURPS quality model is built and extended to be used in the IBM Rational Software Company. Therefore, it is a special-purpose quality model, that is, for the benefits of that company.
4. The metrics in the lower level of the McCall's, Boehm's, Doromey's and FURPS quality models are neither clearly nor completely defined and connected to the upper level of the quality models. For example, in McCall's quality model, the metrics should be clearly and completely defined and connected to the corresponding quality criteria, see Figure 1.

The ISO 9126-1 quality model is the most useful one since it has been build based on an international consensus and agreement from all the country members of the ISO organization.

Corresponding Author:
Dr. Rafa E. Al-Qutaish
Al Ain University of Science and Technology – Abu Dhabi Campus, P.O. Box: 112612, Abu Dhabi, UAE.
E-mail: **rafa@ieee.org**

References

1. Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., McLeod, G., Merritt, M. Characteristics of Software Quality. North Holland Publishing, Amsterdam, The Netherlands, 1978.
2. Boehm, B. W., Brown, J. R., Lipow, M. Quantitative evaluation of software quality. In Proceedings of the 2nd international conference on Software engineering, IEEE Computer Society, Los Alamitos (CA), USA, 1976; 592-605.
3. Dromey, R. G. A model for software product quality. IEEE Transactions on Software Engineering, 1995; 21:146-162.
4. Dromey, R. G. Concerning the Chimera [software quality]. IEEE Software, 1996; 13:33-43.
5. Grady, R. B. Practical Software Metrics for Project Management and Process Improvement. Prentice Hall, Englewood Cliffs, NJ, USA, 1992.
6. IEEE. IEEE Std. 610.12: Standard Glossary of Software Engineering Terminology. The Institute of Electrical and Electronics Engineers, New York, NY, USA, 1990.
7. ISO. ISO/IEC IS 9126: Software Product Evaluation - Quality Characteristics and Guidelines for their Use. International Organization for Standardization, Geneva, Switzerland, 1991.
8. ISO. ISO/IEC 14598-1: Software product evaluation - Part 1: General overview. International Organization for Standardization, Geneva, Switzerland, 1999.
9. ISO. ISO/IEC 9126-1: Software Engineering - Product Quality - Part 1: Quality Model. International Organization for Standardization, Geneva, Switzerland, 2001.
10. ISO. ISO/IEC TR 9126-2: Software Engineering - Product Quality - Part 2: External Metrics. International Organization for Standardization, Geneva, Switzerland, 2003.
11. ISO. ISO/IEC TR 9126-3: Software Engineering - Product Quality - Part 3: Internal Metrics, International Organization for Standardization, Geneva, Switzerland, 2003.
12. ISO. ISO/IEC TR 9126-4: Software Engineering - Product Quality - Part 4: Quality in Use Metrics. International Organization for Standardization, Geneva, Switzerland. Switzerland, 2004.
13. Jacobson, I., Booch, G., Rumbaugh, J. The Unified Software Development Process. Addison Wesley, 1999.
14. Kitchenham, B., Pfleeger, S. L. Software Quality: the Elusive Target. IEEE Software, 1996; 13: 12-21.
15. Kruchten, P. The Rational Unified Process: An Introduction. Addison Wesley, 2000.
16. McCall, J. A., Richards, P. K., Walters, G. F. Factors in Software Quality, Volumes I, II, and III. US Rome Air Development Center Reports, US Department of Commerce, USA, 1977.
17. Pressman, R. S. Software Engineering: A Practitioner's Approach. McGraw-Hill, New York, NY, USA, 2004.
18. Petrasch, R. The Definition of Software Quality: A Practical Approach. In Proceedings of the 10th International Symposium on Software Reliability Engineering, 1999; 33-34.
19. Veenendaal, E. V., McMullan, J. Achieving Software Product Quality, Den Bosch, UTN Publishers, Amsterdam, The Netherlands, 1997.

Submitted on 12/9/2009