Oriental Journal of Computer Science & Technology Vol. **4**(1), 229-236 (2011)

Analysis of Framework on Evaluation of Qualitative Models of Software Development System

ASHISH RASTOGI

Department of Computer Science, GG University, Bilaspur - 495 001 (India).

(Received: April 12, 2011; Accepted: June 04, 2011)

ABSTRACT

As we know that the Software market is growing very fast. The main purpose of the most software producers is produce the software of very high Quality. Software quality is a multi-dimensional content which is easily distinguishable and measurable. Although the Quality of the software is dependent on Functional and Non Functional Requirement of the user. To determine this content more exact, the qualitative models have been presented in which different aspects of this matter are investigated. But the existences of different models and using different expressions have made the comprehension of this content a little hard. In this research paper we try to introduce models and their analytical comparison, determine software qualification and its qualitative characteristics more clearly.

Key words: Software Quality, Functional and Non Functional Requirement, Qualitative Models.

INTRODUCTION

All software developers have a main purpose that is, producing qualitative software. But there is no exact and specified definition of software qualification that can be confirmed by all specialists. Software qualification is not one-dimensional content but is defined by a number of characteristics. Considerable qualitative characteristics in software system are presented in a semi-tree construction called qualitative software model. Presented qualitative models although have many common points, are different in some contents and fields. This research by investigating and comparing the presented models, tries to make a better comprehension of software qualification. This analysis can make or provide an opportunity in order to present new qualitative models especially for software systems which have specific subjects. This paper can provide a situation for stakeholders of software systems to understand content qualification better and to express their qualitative requirements more completely and more exactly. This essay after the introduction, investigate the content qualification in software system.

Related work

Although the term quality seems self-explanatory in everyday usage, in practice there are many different views of what it means and how it should be achieved as part of a software production process.

ISO Definitions

ISO 9000 is concerned with quality assurance to provide confidence that a product will satisfy given requirements.

Interpreted literally, this puts quality in the hands of the person producing the requirements specification a product may be deemed to have quality even if the requirements specification is inappropriate.

This is one of the interpretations of quality reviewed by Garvin (1984). He describes it as *Manufacturing quality:* a product which conforms to specified requirements. A different emphasis is given in ISO 8402 which defines quality as the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs. This is an example of what Garvin calls *Product quality:* an inherent characteristic of the product determined by the presence or absence of measurable product attributes.

In another study examining views of quality, David Garvin studied how quality is perceived in various domains, including philosophy, economics, marketing, and operations management. He concluded that "quality is a complex and multifaceted concept" that can be described from five different perspectives.

- Transcendental View The *transcendental view* sees quality as something that can be recognized but not defined.
- User View The *user view* sees quality as fitness for purpose.
- Manufacturing View The *manufacturing view* sees quality as conformance to specification.
- Product View The *product view* sees quality as tied to inherent characteristics of the product.
- Value based view The *value-based view* sees quality as dependent on the amount a customer is willing to pay for it.

Many organisations would like to be able to identify those attributes which can be designed into a product or evaluated to ensure quality. ISO 9126 (1992) takes this approach, and categorises the attributes of software quality as:
functionality, efficiency, usability, reliability, maintainability and portability.

To the extent that user needs are well-defined and common to the intended users this implies that quality is an inherent attribute of the product. However, if different groups of users have different needs, then they may require different characteristics for a product to have quality for their purposes. Assessment of quality thus becomes dependent on the perception of the user.

Evaluation Criteria

Chen-Burger formalized the rules and guidelines for model constructions, which allowed for a number of automated model critiques. The following types of critiques are provided [CHEN98].

- Correctness: detects structural, syntactic and semantic errors
- Completeness: identifies incomplete information in the model and suggests which missing concepts might need to be included.
- Consistency: points out discrepancies in the model.
- Appropriateness: shows deviations from standard practices.
- Alternatives: searches for similar standard models and presents them as alternatives for a given modelling decision. (This is based on a case library of known models.)

Criteria for Evaluating Methodologies and Related Frameworks

Avison & Fitzgerald [AVIS95:435-437] have an extensive list with areas of concern when comparing methodologies, and suggest a large set of requirements for the design of a methodology. Many of these are based on other authors although some of the original unpublished sources are difficult to obtain. The following extracts only those requirements which could also be considered applicable to models. From Catchpole [CATC87], who summarized the views of a number of other authors in his PhD thesis:

- Documentation easily understandable by user & analysts.
- Separate logical and physical designs.
- Design validity checks for inconsistencies, inaccuracies and incompleteness.
- Easy change.
- Teachability of method.
- Draw/model the boundary between what can/cannot be computerised explicitly.
- Design for future change.
- Effective communication.
- Simplicity.
- Ongoing relevance.
- Automated development aids.
- Consideration of user goals and objectives.
- Systematic way of looking into the future to incorporate possible future changes.
- Integration of technical and non-technical systems.

An Overview of Evaluation Frameworks

The following represents a selection of frameworks for evaluating the quality of modelling approaches and methodologies. Many of these can be partially adapted to evaluate also the outcome

Table 1: Factors are grouped into 3 major categories

Software quality metric / Quality factor	Operation					Revision			Transition		
	Correctness	Reliability	Efficiency	Integrity	Usability	Maintainability	Flexibility	Testability	Portability	Reusability	Interoperability
Auditability				x				x			
Accuracy		x									
Communication commonality											x
Completeness	x										
Complexity		x					x	x			
Conciseness			x			x	x				
Consistency	x	x				x	x				
Data commonality											
Error tolerance		x									x
Execution efficiency			x								
Expandability							x				
Generality							x		x	x	x
Hardware independence									x	x	
Instrumentation				x		x		x			
Modularity		x				x	x	x	x	x	x
Operability			x		x						
Security				x							
Self documentation						x	x	x	x	x	
Simplicity		x				x	x	x			
System independence									x	x	
Traceability	x										
Training					x						

or product of a methodology or modelling process, namely the resultant enterprise model. Many more frameworks have been proposed but there is a high degree of overlap between many of them. The following frameworks are those from which I have drawn my model quality evaluation criteria.

McCall's Quality Factors / The GE model

McCall et al proposed a comprehensive categorization of factors that affect software quality. They distinguish clearly between the quality factor and the quality metric which can be measured directly or indirectly. The following table lists both, as well as the correlation between them [PRES97c; GILL97]. The factors which are grouped into categories i.e revision, transition and operation are summarized in Table 1.

ISO 9126

More recently, international efforts have led to the development of a standard for software-quality measurement, ISO 9126.The standards group has recommended six characteristics to form a basic set of independent quality characteristics (Fig. 1).

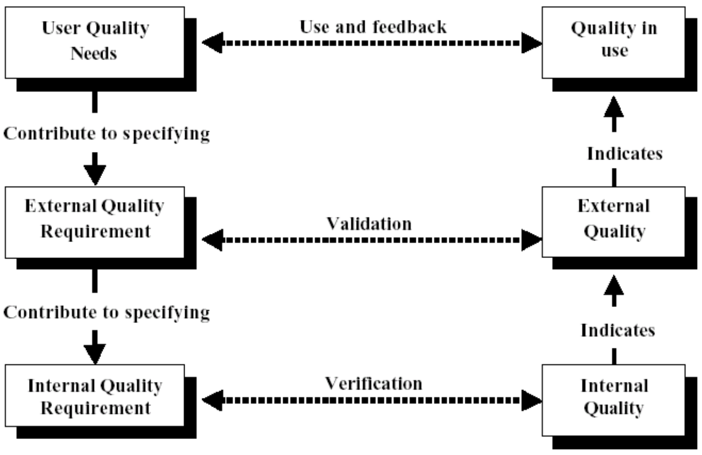

Fig. 1: Quality in the software lifecycle ISO 9126

The Following are the Quality Characteristics of ISO 9126 :

Functionality

A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.

Reliability

A set of attributes that bear on the capability of software to maintain its performance level under stated conditions for a stated period of time.

Usability

A set of attributes that bear on the effort needed for use and on the individual assessment of such use by a stated or implied set of users.

Efficiency

A set of attributes that bear on the relationship between the software's performance and the amount of resources used under stated conditions.

Maintainability

A set of attributes that bear on the effort needed to make specified modifications (which. may include corrections, improvements, or adaptations of software to environmental changes and changes in the requirements and functional specifications).

Portability

A set of attributes that bear on the ability of software to be transferred from one environment to another (this includes the organizational, hardware or Software environment).

The standard also includes a sample quality model that refines the features of ISO 9126 into several subcharacteristics. The standard recommends measuring the characteristics directly, but does not indicate clearly how to do it. Rather, the standard suggests that if the characteristic cannot be measured directly (particularly during development), some other related attribute should be measured as a surrogate to predict the required characteristic. However, no guidelines for establishing a good prediction system are provided.

Although the ISO 9126 model is similar to McCall's, there are several differences. Clearly, the ISO model uses a different quality framework and termi- nology, and the term "quality characteristic" is used instead of quality factor. The other elements of the ISO framework (as defined in associated guidelines) are: quality subcharacteristics to refine the characteristic, indicators to measure quality subcharacteristics, and data elements to construct an indicator. (Indicators are usually ratios derived from data elements. For example, the fault rate can be defined as the ratio of number of faults to product size.) In addition to the different terminology, there are structural differences between the models. Unlike earlier American models, the ISO framework is completely hierarchical each subcharacteristic is related to only one characteristic. Also, the subcharacteristics relate to quality aspects that are visible to the user, rather than to internal software properties. Thus, the ISO model reflects more of a user view, while the McCall model reflects more of a product view.

The Böhm model

A fairly similar model was proposed by Böhm, which contains very much the same characteristics, although there are a number of semantic nuances. The model has two levels, with the intermediate level further split into primitive characteristics which can be measured [BÖHM76].

Note that the emphasis of Böhm's model is, similarly to McCall's, on the more technical criteria. Hyatt summarizes both of the above and maps their criteria against those mentioned in the ISO 9126 standard, which defines and describes the software product quality criteria more formally [HYAT96].

Table 2: Software Quality Models [HYAT96]

Criteria/Goals	McCall, 1977	Boehm, 1978	ISO 9126, 1993
Correctness	X	X	Maintainability
Reliability	X	X	X
Integrity	X	X	
Usability	X	X	X
Efficiency	X	X	X
Maintainabiilty	X	X	X
TestabilitY	X		Maintainability
Interoperability	X	X	
Flexibility	X	X	
Reusability	X	X	
Portability	X	X	X
Clarity		X	
Modifiability		X	Maintainability
Documentation		X	
Resilience		X	
Understandability		X	Maintainability
Validity			X
Functionality			
Generality		X	
Economy		X	

He uses these to build his own quality framework. Very interesting are his suggested metrics for measuring some of these quality attributes. Two of these are also used in this research: document structure and readability index.

IEEE Model

IEEE Institute, in fact, has given a scale and standard to provide a qualitative model and has not construction to show qualitative model and emphasizes on how to design the measurement ways of qualitative factors. This suggested construction is semi-tree and has model it is allowed to define metrics for any factors if there is a direct measurability after the first level. For the first as follows:

Efficiency
Temporal economy and resource economy.
Function
Completeness, accuracy, security, compatibility and interoperability.
Supportability
Testability, extensibility and correctability.
Portability
Independency from hardware, independency from software.
Reliability
Nondeficiency, error tolerance and availability
Usability
Comprehensibility, ease of learning, usability,

Avison & Fitzgerald's Framework for Methodology Comparison

Avison & Fitzgerald [AVIS95:446-448] have developed the following framework for comparing methodologies, based on a number of previous attempts and other authors such as Wood-Harper. Although their approach resembles a hierarchical structure, the authors describe it as a framework because it takes contextual and philosophical considerations into account. These considerations include academic methodology taxonomies, and the actual evaluation criteria for each element depend on the methodology under consideration. Each element is elaborated on in much detail in the text. Their seven basic framework elements, with sub-elements, are:

1. Philosophy
   a. Paradigm: (hard) science versus (soft) systems (see also [HIRS89]).
   b. Objectives.
   c. Domain.
   d. Target: particular types/sizes of organisations?
2. Model (verbal / analytical / iconic / simulation).
3. Techniques and tools.
4. Scope (life cycle, level of detail).
5. Outputs.
6. Practice:
   a. Background: academic or practioner/ commercial.
   b. User base: numbers & types of users.
   c. Players: users and/or analyst.
7. Product: Does it include: software? Documentation? Training? Telephonic/online help?

They also suggest two possible additional elements:
´ Quantity of specifications and documentation.
´ User modifiability.

Problems with the Current Frameworks

There are a number of problems with the application of the frameworks mentioned earlier. The three major overall problems are:

The Grounding Problem

All lack a fundamental conceptual or theoretical basis: the distinctions seem to be ad-hoc, based on a natural grouping or structuring of the desired factors but there is no underlying theoretical or philosophical basis for the framework dimensions. Refer to [FRAN99b] for the importance of a strong theoretical grounding for an evaluation framework instead of an ad-hoc approach.

The Partial Applicability Problem

Most of them apply only partially to the evaluation of generic enterprise models. A lot of criteria relate to the development process, which is not applicable in our case.

The Lack of Generality Problem

None of the frameworks is generic in such a way that they could be applied to evaluate the quality of similar "intellectual works" in IS or any other discipline i.e. there is no relationship between these frameworks (say for measuring the quality of software) and any possible framework to evaluating, say, models, systems architectures, products, or artworks. This is partly due to their lack of theoretical grounding. But even so, one would expect a quality evaluation framework to have some applicability or transferability to related domains.

Thus the challenge is to develop a framework for evaluating enterprise models that has sound theoretical foundations, is specific to the evaluation of enterprise models yet can be ported to similar domain or problem areas.

Our approach to Software Quality Model Evaluation

To highlight some solutions of the above-mentioned problems, we deal with the 9 steps needed to apply our approach to software quality evaluation, which solves some of the open issues.

Step by Step

The following steps highlight the main ideas to implement software quality assessment while considering human requirements.

Step 1

Choosing Category of People. We must choose at least a person from the category of people, hich our software evaluation will be implemented for, for example: Programmers, End-user.

Step 2

Identifying Sample Program. We must choose a simple program (BP) to be considered as sample evaluation set of our model.

Step 3

Building a Quality Model. The process of building a quality model decomposes in two main tasks generally : Choosing a super-characteristic and Choosing and organizing characteristics related to super characteristic[10]. In our case study, we consider design patterns especially as bridge between internal attributes of programs, external quality characteristics, and software engineers.

Step 4

Human Evaluation. The small group, or at least one person from the group, must look in the program or product BP and evaluate the quality characteristics we defined in our quality model, the evaluation could be in form of numerical value or different levels on a Lickert scale.

Step 5

Computing Software Metrics over BP. By using software metrics we evaluate BP numerical values related to software internal attributes.

Step 6

Machine Learning Tools.

Step 7

Computing Software Metrics over EP. Software metrics are used to assess the values of internal attributes over the EP in the same way as they were for the evaluation of BP

Step 8

Adapting Metric.

CONCLUSION

There are various approaches available for analyzing the framework for quality models. Our approach is different from all other available models. Some more experiments are needed to implement it.

REFERENCES

1. Pressman, R.S., Software Engineering: A Practitioner' approach. McGraw-hill (2000).
2. Cavano, J.P. and J.A. McCall, A Framework for the Measurement of Software Quality. Procs. ACM Software Quality Assurance Workshop, pp: 133-139 (1978).
3. Bevan, N., Quality in use: Meeting user needs for quality. *Journal of System and Software*, Elsevier., **49**(1): 89-96 (1999).
4. ISO/IEC 9126, Information Technologysoftware Product Evaluation: Quality Characteristics and Guideline for

Their Use (1991).

5. Khosravi, K. and Y. Gueheneuc, A Quality Model for Design Patterns, M. S. Thesis, Laboratory of Open Distributed Systems and Software Engineering, Dept. of Informatics and Operations Research, University of Montreal (2004).
6. Kazman, R., L. Bass and P. Clements, 2003. Software Architecture in Practice 2Ed. Addison Wesley. Klein, M., P. Clements and R. Kazman (2002).
7. Evaluating Software Architectures: Methods and Case Studies. Addison Wesley.
8. IEEE, IEEE Standard for a Software Quality Metrics Methodology, IEEE Std 1061-1992, IEEE Computer Society. Quality. Elsevier North-Holland. Describe and Evaluate Software Architecture. Rev (1993).
9. R. Arnold and S. Bohner. Software Change Impact Analysis. IEEE Computer Society Press. ISBN 0-8186-7384-2 (1996).
10. V. R. Basili, R. W. Selby, and D. H. Hutchens. Experimentation in Software Engineering. *IEEE Transactions on Software Engineering,* **12**(7): 733-743 (1986).
11. V. Basili, L. Briand, and W. Melo. A Validation of Object-oriented Design Metrics as Quality Indicators. *IEEE Transactions on Software Engineering*, **22**(10): 751-761 (1996).
12. L. Briand, J. Wuest, and H. Lounis. Using Coupling Measurement for Impact Analysis in Object-Oriented System. IEEE International Conference on Software Maintenance (ICSM), Oxford, UK (1999).
13. R. Dromey. A Model for Software Product Quality. IEEE Transactions on Software Engineering **21**: 146-62 (1995).
14. R. Dromey. Cornering the Chimera. IEEE Software **13**: 33-43 (1996).
15. N. Fenton and S. Pfleeger. Software Metrics: A Rigorous and Practical Approach. Int'l Thomson Computer Press (1996).
16. International Standards Organizations. Information Technology- Software Product Evaluation- Quality Characteristics and Guidelines for their Use, ISO/IEC IS 9126, Geneva, Switzerland (1991).
17. J. McCall, P. Richards, and G. Walters. Factors in Software Quality. RADC TR-77-369 1977. US Rome Air Development Center Reports NTIS AD/A-049 014,015,055 (1977).
18. IEEE Computer Society. Proceedings Sixth International Software Metrics Symposium, Boca Raton, Florida, USA. IEEE Computer Society Press (1999).
19. S. Morasca and L. C. Briand. Towards a Theoretical Framework for Measuring Software Attributes. In Proceedings of the 4th International Software Metrics Conference, pp 119-126, Albuquerque, New Mexico, (1997).
20. R. Offen and R. Jeffery. Establishing Software Measurement Programs. *IEEE Software,* **14**(2):45-53 (1997).
21. J. Poulin. Measuring Software Reuse-Principles, Practice, and Economic models. Addison Wesley (1997).
22. M. Price and S. Demurjian. Analyzing and Measuring Reusability in Object-Oriented Designs. In Proceedings of OOPSLA'97, p22, Atlanta Georgia USA, (1997).