Software Quality Model Requirements for Software Quality Engineering

Marc-Alexis Côté[1], Witold Suryn[2], Elli Georgiadou[3]

[1] Ubisoft Canada, Quebec, Canada.

Email: marc-alexis.cote@ubisoft.com

[2] Software and Information Technology Engineering Dept, École de technologie supérieure. Montréal, Canada.

E-mail: witold.suryn@etsmtl.ca wsuryn@ele.etsmtl.ca

[3] Middlesex University, London, UK

Email: e.georgiadou@mdx.ac.uk

Abstract

Software Quality Engineering is an emerging discipline that is concerned with improving the approach to software quality. It is important that this discipline be firmly rooted in a quality model satisfying its needs. In order to define the needs of this discipline, the meaning of quality is broadly defined by reviewing the literature on the subject. Software Quality Engineering needs a quality model that is usable throughout the software lifecycle and that it embraces all the perspectives of quality. The goal of this paper is to propose a quality model suitable for such a purpose, through the comparative evaluation of existing quality models and their respective support for Software Quality Engineering.

Introduction

Over the last decade, the general focus of the software industry has shifted from providing ever more functionality to improving what has been coined as the user experience. The user experience refers to characteristics such as ease-of-use, security, stability and reliability. Improvements in such areas lead to an improved quality as perceived by the end users. Some software products, most notably Microsoft's next iteration of their Windows operating system, have been delayed by as much as two years in order to improve their quality. There is no doubt that software quality is becoming an increasingly important subject in software engineering.

Traditionally, software requirements have been classified either as functional or non-functional with eventual notions of quality hidden in the latter. As the industry focus is shifting from functionality to improving quality, a new category of requirements focused on quality is emerging. In order to define these new quality requirements, quality itself must be defined. A quality model provides the framework towards a definition of quality. Engineers have long recognised that in order for something to find its way in a product, it should be properly defined and specified. Unfortunately, the push towards software quality that can be observed in the industry today is lacking a solid foundation in the form of an agreed upon quality model that can be used not only to evaluate software quality, but also to specify it.

Bourque [7] suggests that the implementation of quality in a software product is an effort that should be formally managed throughout the Software Engineering lifecycle. The implementation of quality should therefore begin with the specification of user quality requirements. Such an approach to the implementation of quality leads to Software Quality Engineering. Suryn [37] has suggested that this discipline be defined as *the application of a continuous, systematic, disciplined, quantifiable approach to the development and maintenance of quality of software products and systems; that is, the application of quality engineering to software*.

The objective of this paper is to identify the requirements for a software quality model to be used as a foundation to Software Quality Engineering.

Definition of Software Quality

What exactly constitutes the quality of a product is often the subject of a hot debate. The reason the concept of quality is so controversial is that people fail to agree on what it means. For some it is "[the] degree to which a set of inherent characteristics fulfills requirements" [22] while for others it can be synonymous with "customer value" or even "defect levels" [19]. A possible explanation as to why any of these definitions fail to garner a consensus is that they generally fail to recognize the different perspectives of quality. Kitchenham and Pfleeger [28], by reporting the teachings of David Garvin, report on the 5 different perspectives of quality:

- The transcendental perspective deals with the metaphysical aspect of quality. In this view of quality, it is "something toward which we strive as an ideal, but may never implement completely." [28];
- The user perspective is concerned with the appropriateness of the product for a given context of use. Kitchenham and Pfleeger further note that "whereas the transcendental view is ethereal, the user view is more concrete, grounded in the product characteristics that meet user's needs";
- The manufacturing perspective represents quality as conformance to requirements. This aspect of quality is stressed by standards such as ISO

9001, which defines quality as "[the] degree to which a set of inherent characteristics fulfills requirements" [22]. Other models, like the Capability Maturity Model (CMM) state that the quality of a product is directly related to the quality of the engineering process, thus emphasising the need for a manufacturing-like process;

- The product perspective implies that quality can be appreciated by measuring the inherent characteristics of the product. Such an approach often leads to a bottom-up approach to software quality: by measuring some attributes of the different components composing a software product, a conclusion can be drawn as to the quality of the end product;
- The final perspective of quality is value-based. This perspective recognises that the different perspectives of quality may have a different importance, or value, to various stakeholders.

One could argue that in a world where conformance to ISO and IEEE standards is increasingly present in contractual agreements and used as a marketing tool [1], all the perspectives of quality are subordinate to the manufacturing view. This importance of the manufacturing perspective has increased throughout the years through works like Quality is Free (Crosby, 1979) and the popularity of movements like Six-Sigma [3]. The predominance of the manufacturing view in Software Engineering can be traced back to the 1960s, when the US Department of Defense and IBM gave birth to Software Quality Assurance [38]. This has led to the belief that adherence to a development process, as in manufacturing, will lead to a quality product. The corollary to this belief is that process improvement will lead to improved product quality. According to many renowned researchers, this belief is false, or at least flawed. Geoff Dromey states:

> *"The flaw in this approach [that you need a quality process to produce a quality product] is that the emphasis on process usually comes at the expense of constructing, refining, and using adequate product quality models."[14].*

Kitchenham and Pfleeger reinforce this opinion by stating:

> *"There is little evidence that conformance to process standards guarantees good products. In fact, the critics of this view suggest that process standards guarantee only uniformity of output [...]"[28].*

Furthermore, data available from Agile [19] projects show that high quality is attainable without following a manufacturing-like approach.

However, recent studies conducted at Motorola [16, 12] and Raytheon [16] show that there is indeed a correlation between the maturity level of an organization as measured by the Capability Maturity Model and the quality of the resulting product. These studies provide data on how a higher maturity level (as measured by the CMM) can lead to:

- Improved error/defect density (i.e. the error/defect density lowers as maturity improves)
- Lower error rate

- Lower cycle time (time to complete parts of the lifecycle)
- Better estimation capability

From these results, one could conclude quality can be improved by following a mature process. Georgiadou (2003a) studied the development of lifecycle models, and established that the maturity of the development process is reflected by the emphasis and location of testing and other quality assurance activities. Her study demonstrated that the more mature the process and its underlying lifecycle model the earlier the identification of errors in the deliverables. However, these measured improvements are directly related to the manufacturing perspective of quality. Therefore, such quality improvement efforts fail to address the other perspectives of quality. This might be one of the reasons that some observers of the software development scene perceive the "quality problem" as one of the main failings of the software engineering industry. Furthermore, studies show that improvement efforts grounded in the manufacturing perspective of quality are difficult to scale down to smaller projects and/or smaller teams [29, 4]. Indeed, rather than being scaled down in smaller projects, these practices are simply not performed.

Over recent years, researchers have proposed new models that try to encompass more perspectives of quality than just the manufacturing view. Geoff Dromey [13,14] proposed such a model in which the quality of the end product is directly related to the quality of the artifacts that are a by-product of the process being followed. Therefore, he developed different models that can be used to evaluate the quality of the requirements model, the design model and the resulting software. The reasoning is that if quality artifacts are conceived and produced throughout the lifecycle, then the end product will manifest attributes of good quality. This approach can clearly be linked to the product perspective of quality with elements from the manufacturing view. This is certainly a step forward from the manufacturing-only approach described above, but it fails to view the engineering of quality as a process that covers all the perspectives of quality. Pfleeger [33] warns against approaches that focus only on the product perspective of quality:

> *"This view [the product view] is the one often advocated by software metrics experts; they assume that good internal quality indicators will lead to good external ones, such as reliability and maintainability. However, more research is needed to verify these assumptions and to determine which aspects of quality affect the actual product's use."*

Georgiadou [18] developed a generic, customisable quality model (GEQUAMO) which enables any stakeholder to construct their own model depending on their requirements. In a further attempt to differentiate between stakeholders Siaka et al [35] studied the viewpoints of users, sponsors and developers as three important constituencies/stakeholders and suggested attributes of interest to each constituency as well as level of interest. More recently, Siaka and Georgiadou [36] reported the results of a survey amongst practitioners (from the UK, Greece, Egypt and Cyprus) on the importance placed on product quality characteristics. Using their empirical results they extended ISO 9126 by adding two new characteristics

namely Extensibility and Security which have gained in importance in today's global and inter-connected environment.

The above observations illustrate the main disagreements that exist in both the research community and the industry on the subject of software quality. The goal of a quality model is in essence to provide an operational definition of quality. While specific definitions have been established for given contexts, there is no consensus as to what constitutes quality in the general sense in software engineering. A first requirement for a software quality model to be useful as a foundation for Software Quality Engineering is thus to encompass all the perspectives of quality mentioned at the beginning of this section.

Specification and evaluation of quality

Software Quality Engineering calls for a formal management of quality throughout the lifecycle. In order to support this requirement, a quality model should have the ability to support both the definition of quality requirements and their subsequent evaluation. This can be explained by referring to the manufacturing perspective of quality, which states that quality is conformance to requirements. A quality model that is to be used as the foundation for the definition of quality requirements should help in both the specification of quality requirements and the evaluation of software quality.

IEEE Std 1061-1998 [20] defines this as a top to bottom and bottom to top approach to quality:

From a top down perspective the [quality] framework facilitates:

- *Establishment of quality requirements factors, by customers and managers early in a system's life cycle;*
- *Communication of the established quality factors, in terms of quality sub-factors, to the technical personnel;*
- *Identification of metrics*[1] *that are related to the established quality factors and quality sub-factors.*

From a bottom up perspective the [quality] framework enables the managerial and technical personnel to obtain feedback by

- *Evaluating the software products and processes at the metrics level;*
- *Analysing the metric values to estimate and assess the quality factors.*

---

[1] In 2002, the ISO/IEC JTC1 sub-committee SC7 – Systems and Software Engineering – replaced the term "metric" by "measure" to align its vocabulary with the one used in metrology. This paper will use the term measure whenever possible.

A quality model that is to be used as the foundation for the definition of quality requirements should help in both the specification of quality requirements and the evaluation of software quality. In other words, it should be usable from the top of the development process to the bottom and from the bottom to the top.

Evaluation of quality models

Three requirements that a quality model should possess to be a foundation for Software Quality Engineering have been identified:

- A quality model should support the 5 different perspectives of quality as defined by Kitchenham and Pfleeger [28];
- A quality model should be usable from the *top to the bottom* of the lifecycle as defined by IEEE Std 1061-1998 [20], i.e. should allow for defining quality requirements and their further decomposition into appropriate quality characteristics, subcharacteristics and measures;
- A quality model should be usable from the *bottom to top* of the lifecycle as defined by IEEE Std 1061-1998 [20], i.e. should allow for required measurements and subsequent aggregation and evaluation of obtained results.

Four quality models will be evaluated with respect to these requirements.

McCall

McCall [32] introduced his quality model in 1977. According to Pfleeger [33], it was one of the first published quality models. Figure 1 presents this quality model. Each quality factor on the left hand side of the figure represents an aspect of quality that is not directly measurable. On the right hand side are the measurable properties that can be evaluated in order to quantify the quality in terms of the factors. McCall proposes a subjective grading scheme ranging from 0 (low) to 10 (high).

Regarding this model, Pressman notes that "unfortunately, many of the metrics defined by McCall et al. can be measured only subjectively" [34]. It is therefore difficult to use this framework to set precise and specific quality requirements. Furthermore, some of the factors and measurable properties, like traceability and self-documentation among others, are not really definable or even meaningful at an early stage for non-technical stakeholders. This model is not applicable with respect to the criteria outlined in the IEEE Standard for a Software Quality Metrics Methodology for a top to bottom approach to quality engineering. Furthermore, it emphasises the product perspective of quality to the detriment of the other perspectives. It is therefore not suited as a foundation for Software Quality Engineering according to the stated premises.

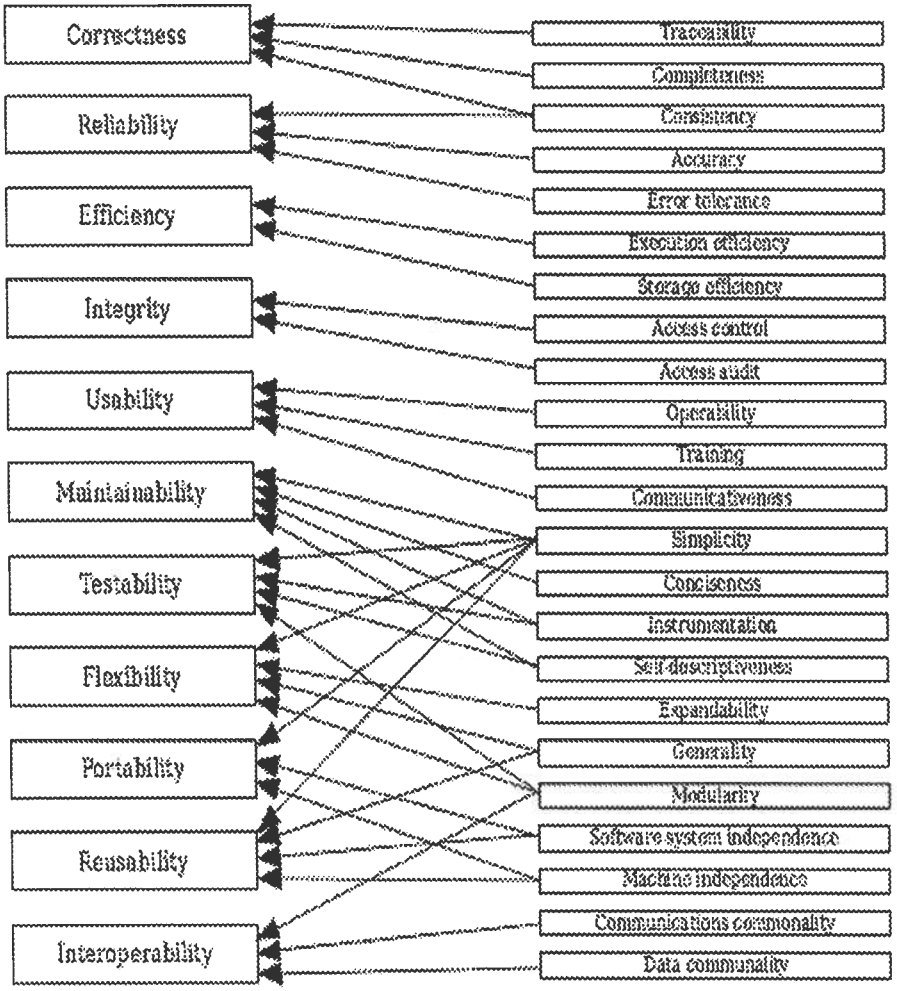

Figure 1: McCall's Quality Model
Adapted from Pfleeger (2003) and McCall et al. (1977)

Boehm

Boehm's quality model improves upon the work of McCall and his colleagues [5]. As Figure 2 shows, this quality model loosely retains the factor-measurable property arrangement. However, for Boehm and his colleagues, the prime characteristic of quality is what they define as "general utility". According to Pfleeger (2001), this is an assertion that first and foremost, a software system must be useful to be considered a quality system. For Boehm, general utility is composed of as-is utility, maintainability and portability [6]:

- How well (easily, reliably, efficiently) can I use it [software system] as-is?

- How easy is it to maintain (understand, modify, and retest)?
- Can I still use it if I change my environment?

If the semantics of McCall's model are used as a reference, the quality factors could be defined as: Portability, Reliability, Efficiency, Human Engineering, Testability, Understandability and Modifiability. These factors can be decomposed into measurable properties such as Device Independence, Accuracy, Completeness, etc. Portability is somewhat incoherent in this classification as it acts both as a top level component of general utility, and as a factor that possesses measurable attributes.

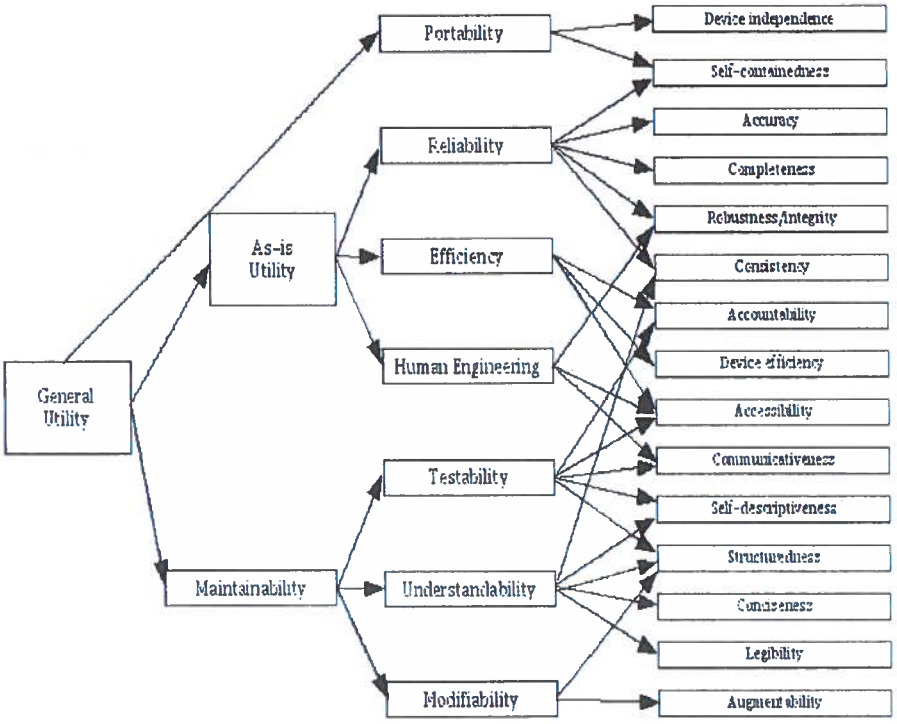

Figure 2: Boehm's quality model
Adapted from Pfleeger (2003), Boehm et al. (1976; 1978)

It is interesting to note that in opposition to McCall's model, Boehm's model is decomposed in a hierarchy that at the top addresses the concerns of end-users while the bottom is of interest to technically inclined personnel. It is in effect the emergence of the user perspective of quality. However, this interest wanes when one reads Boehm's definition of the characteristics of software quality. Except for *General Utility and As-is Utility*, all definitions begin with "Code possesses the characteristic [...]". The measurable properties and characteristics therefore concentrate on highly technical details of quality that are difficult to grasp for non-technical stakeholders that are typically involved early in the software lifecycle.

The characteristics *General Utility* and *As-is Utility* are too generic and imprecise to be useful for defining verifiable requirements. Like the McCall model, this model is mostly useful for a bottom to top approach to software quality (i.e. it can effectively be used to define measures of software quality, but is more difficult to use to specify quality requirements).

While this model is a step forward in the sense that it provides basic support for a top to bottom approach to software quality, this support is too ephemeral to be considered as a solid foundation for quality engineering.

Dromey

Dromey's [13] model takes a different approach to software quality than the two previously presented models. For Dromey, a quality model should clearly be based upon the product perspective of quality:

> *"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."[13] (Emphasis added to support the argument)*

Dromey has built a quality evaluation framework that analyzes the quality of software *components* through the measurement of tangible quality properties (Figure 3). Each artifact produced in the software lifecycle can be associated with a quality evaluation model. Dromey gives the following examples of what he means by software components for each of the different models:

- Variables, functions, statements, etc. can be considered components of the implementation model;
- A requirement can be considered a component of the requirements model;
- A module can be considered a component of the design model;
- Etc.

According to Dromey [13], these components all possess intrinsic properties that can be classified into four categories:

- Correctness: Evaluates if some basic principles are violated.
- Internal: Measure how well a component has been deployed according to its intended use.
- Contextual: Deals with the external influences by and on the use of a component.
- Descriptive: Measure the descriptiveness of a component (for example, does it have a meaningful name?).

These properties are used to evaluate the quality of the components. This is illustrated in Figure 4 for a variable component present in the implementation model.

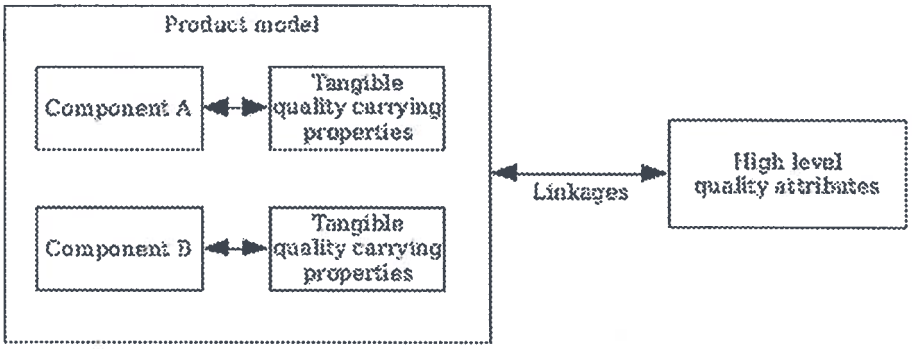

Figure 3 : Dromey's Quality Model

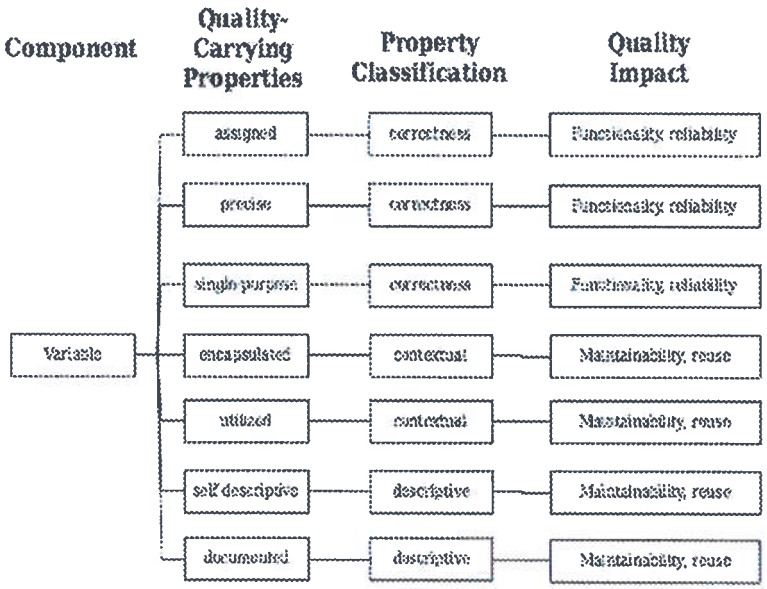

Figure 4 : Quality evaluation of a variable component

It seems obvious from the inspection of the previous figures that Dromey's model is focused on the minute details of quality. This is stated explicitly:

"What we can do is identify and build in a consistent, harmonious, and complete set of product properties (such as modules without side effects) that result in manifestations of reliability and maintainability." [13].

For Dromey, the high level characteristics of quality will manifest themselves if the components of the software product, from the individual requirements to the programming language variables1, exhibit quality-carrying properties. Dromey's hypothesis should be questioned. If all the components of all the artifacts produced during the software lifecycle exhibit quality-carrying properties, will the resulting product manifest characteristics such as maintainability, functionality, and others?

The following analogy will be useful in answering this question:

If you buy the highest quality flour, along with the highest quality apples and the highest quality cinnamon, will you automatically produce an apple pie that is of the highest quality?

The answer is obviously negative. In addition to quality ingredients, at least three more things are needed in order to produce an apple pie of the highest quality:

- A recipe (i.e. an overall architecture and an execution process). Dromey acknowledges this by identifying process maturity as a desirable high level characteristic. However, it is only briefly mentioned in both his publications on the subject [13, 14].
- The consumer's tastes must be taken into account. In order for the result to be considered of the highest quality by the consumer, it needs to be tuned to his tastes. This is akin to what is commonly called user needs in software engineering. User needs are completely ignored by Dromey. However, as it was demonstrated in the introduction, they are an integral and non-negligible part of software quality.
- Someone with the qualifications and the tools to properly execute the recipe.

While Dromey's work is interesting from a technically inclined stakeholder's perspective, it is difficult to see how it could be used at the beginning of the lifecycle to determine user quality needs. Dromey [13] states that software quality "must be considered in a systematic and structured way, from the tangible to the intangible". By focusing too much on the tangible, Dromey fails to build a model that is meaningful for stakeholders typically involved at the beginning of the lifecycle. Do end users care about the variable naming convention or module coupling? In most cases, it is doubtful that this question can be answered affirmatively. Therefore, this model is rather unwieldy to specify user quality needs. This does not mean that it cannot be useful later on as a checklist for ensuring that product quality is up to standards. It can definitely be classified as a bottom to top approach to software quality.

Furthermore, as was illustrated at the beginning of this section, this quality model has its roots in the product perspective of quality, to the detriment of other

perspectives. Therefore, it fails to qualify as a foundation for Software Quality Engineering according to the established requirements.

ISO/IEC 9126

In 1991, the International Organization for Standardization introduced a standard named ISO/IEC 9126 (1991): Software product evaluation - Quality characteristics and guidelines for their use. This standard aimed to define a quality model for software and a set of guidelines for measuring the characteristics associated with it. ISO/IEC 9126 quickly gained notoriety with IT specialists in Europe as the best way to interpret and measure quality [2]. However, Pfleeger [33] reports some important problems associated with the first release of ISO/IEC 9126:

- There are no guidelines on how to provide an overall assessment of quality.
- There are no indications on how to perform the measurements of the quality characteristics.
- Rather than focusing on the user view of software, the model's characteristics reflect a developer's view of software.

According to Pfleeger, this first incarnation of ISO/IEC 9126 is not usable as a bottom up approach to quality engineering, and even less usable as a top down approach.

In order to address these concerns, an ISO committee began working on a revision of the standard. The results of this effort are the introduction of a revised version of ISO/IEC 9126 focusing on the quality model, and a new standard, ISO/IEC 14598 [21] focusing on software product evaluation. ISO/IEC 14598 addresses Pfleeger's first concern while the revision to ISO/IEC 9126 aims to resolve the second and third issues. ISO/IEC 9126 is now a four part standard:

- ISO/IEC 9126-1 [24] defines an updated quality model.
- ISO/IEC 9126-2 [26] defines a set of external measures.
- ISO/IEC 9126-3 [27] defines a set of internal measures.
- ISO/IEC 9126-4 [25] defines a set of quality in use measures.

The new quality model defined in ISO/IEC 9126-1 recognises three aspects of software quality and defines them as follows: (the full definition is given as it is pertinent to the discussion that ensues).

- quality in use:

Quality in use is the user's view of the quality of the software product when it is used in a specific environment and a specific context of use. It measures the extent to which users can achieve their goals in a particular environment, rather than measuring the properties of the software itself. [24]

- external quality:

External quality is the totality of characteristics of the software product from an external view. It is the quality when the software is executed, which is typically measured and evaluated while testing in a simulated environment with simulated data using external metrics. During testing, most faults should be discovered and eliminated. However, some faults may still remain after testing. As it is difficult to correct the software architecture or other fundamental design aspects of the software, the fundamental design remains unchanged throughout the testing. [24]

- internal quality:

Internal quality is the totality of characteristics of the software product from an internal view. Internal quality is measured and evaluated against the internal quality requirements. Details of software product quality can be improved during code implementation, reviewing and testing, but the fundamental nature of the software product quality represented by the internal quality remains unchanged unless redesigned. [24]

The internal and external quality model is inspired from McCall and Boehm's work. It is a three-layer model composed of quality characteristics, quality subcharacteristics and quality measures. Figure 5 illustrates this model. More than 100 measures of internal and external quality are proposed as part of the standard. It is important to note that the measures do not make an exhaustive set, which means that other measures can also be used.

Finally, Quality in use is modeled in a different way than internal and external quality. Figure 6 illustrates the two-layer quality in use model composed of characteristics and quality measures.

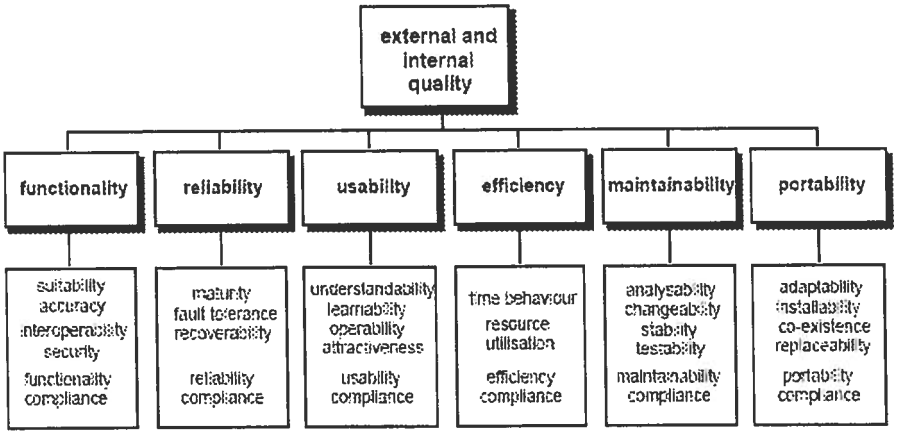

Figure 5 : 3-layer model for internal and external quality. Adapted from [24]

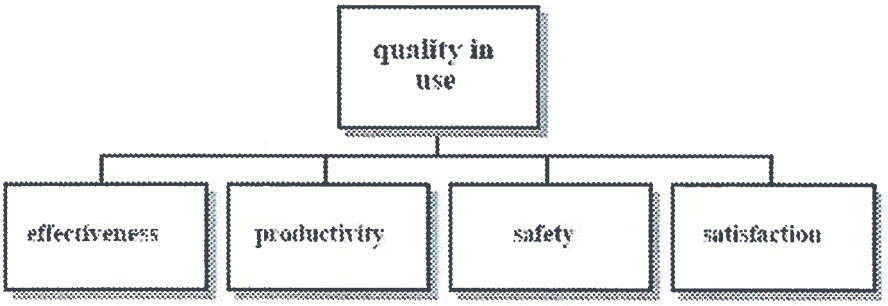

Figure 6 : Quality in use model. Adapted from [24]

Theoretically, internal quality, external quality and quality in use are linked together with a predictive model. This is illustrated in Figure 7.

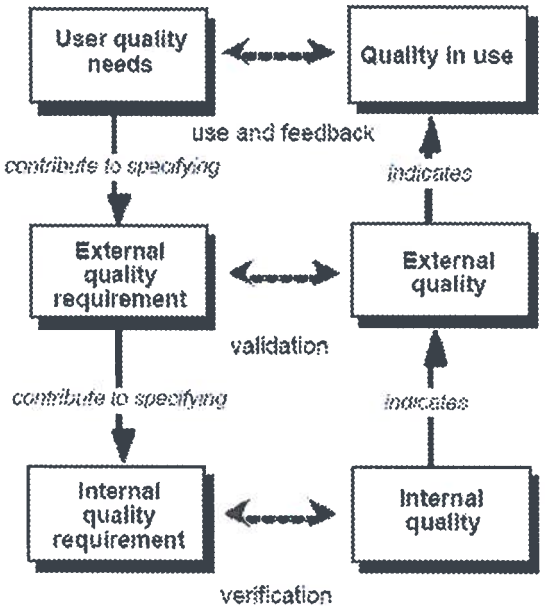

Figure 7 : Relationships between the different aspects of quality. Adapted from [24]

This prediction relationship states that user quality needs should first be established and specified using the Quality in use model. From these requirements as well as other sources, external quality requirements should be established using the external quality model. Finally, the internal quality requirements should be constructed from the external quality requirements and other sources. Once the requirements are established and software construction is under way, the quality model can be used to predict the overall quality. For example, measurement of

internal quality can be useful in predicting external quality. Likewise, measurement of external quality can be useful in predicting quality in use.

The above paragraphs describe the ideal theoretical model that links these three aspects of quality. However, in reality, no model may claim to follow perfectly this prediction mechanism. Although the ISO/IEC 9126 model follows this approach closely, no claims are made as to the real predictive power of the model. While the links between internal and external quality seem rather obvious because the models are essentially the same, caution must be exercised. While the name of the characteristics and subcharacteristics are the same, the links between internal and external quality must be verified empirically. The same reasoning applies to the links between external quality and quality in use.

The new version of ISO/IEC 9126 is gaining momentum in the industry. Some corporate quality models, for example MITRE's SQAE [31], are beginning a migration from a model based on McCall's and Boehm's research to one based on ISO/IEC 9126 [8, 9, 10]. This new version of ISO/IEC 9126 is thus seen as an improvement upon the older quality models.

It is interesting to see how the three aspects of quality defined above can be directly linked to the perspectives of quality that were outlined previously. More specifically:

- ISO/IEC 9126-4, which defines quality in use, is directly related to the user and value-based perspectives. The definition of the user perspective of quality states that it is concerned with the appropriateness of a product for a given context of use. Quality in use is defined as the capability of the software product to enable specified users to achieve specified goals in specified contexts of use. The relationship between the two is clear. Quality in use and the value based perspective of quality are linked essentially through the Satisfaction characteristic. This characteristic inherently recognises that quality can have a different meaning and/or value for different stakeholders. Satisfaction levels can thus be set according to those levels of perception. This has been demonstrated by the study reported in [36].
- ISO/IEC 9126-3, which defines internal quality, and ISO/IEC 9126-2, which defines external quality, are directly related to both the manufacturing and product perspectives. The definitions of the quality characteristics Functionality and Reliability can be linked with the manufacturing perspective of quality. Reliability, Usability, Efficiency, Maintainability and Portability are all inherent characteristics of the product and a manifestation of the product perspective of quality.

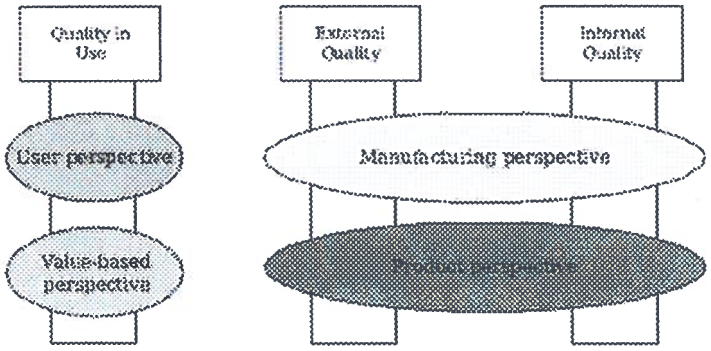

Figure 8 : Relationships between ISO/IEC 9126 and the perspectives of quality

From the review of the different quality models, one might point out that none seem to address the transcendental perspective of quality. One might even ask the following pertinent question: Does ISO/IEC 9126 address the transcendental perspective of quality? Recall that the transcendental perspective of quality relates to quality as something that is recognised but not defined. At this point, the following hypothesis will be made:

> *As the transcendental perspective of quality cannot be defined, it cannot be explicitly implemented in a software product. However, the transcendental aspect of quality will emerge when a holistic approach to quality engineering is taken.*

This model seems to recognise all the perspectives of quality as important contributors to the overall assessment of quality. It takes an incremental approach to software quality that begins with quality in use, something that is easy to grasp for non-technical stakeholders, and ends with internal quality, something more technically inclined stakeholders will feel more comfortable with. Furthermore, there is a comprehensive set of suggested measures that allow for the assessment of software quality.

ISO/IEC 9126 is thus the only model that fulfills all the stated requirements for a model to be useful as a foundation to Software Quality Engineering.

Conclusion

This paper has defined three requirements that a quality model should meet to serve as a foundation to Software Quality Engineering:

- A quality model should support the 5 different perspectives of quality as defined by Kitchenham and Pfleeger [28].

- A quality model should be usable from the *top to the bottom* of the lifecycle as defined by IEEE Std 1061-1998 [20].
- A quality model should be usable from the *bottom to top* of the lifecycle as defined by IEEE Std 1061-1998 [20].

These criteria were applied to four quality models:

- It was found that the models proposed by McCall, Boehm and Dromey focus on the product perspective of quality to the detriment of other perspectives. Furthermore, they are primarily useful in a bottom up approach to quality that is not suitable for Software Quality Engineering.
- ISO/IEC 9126 is the only model that supports all the perspectives of quality (with the exception of the transcendental perspective as noted). Furthermore, its predictive framework clearly supports both the top down and bottom up approaches.

This paper has focused on analysing the semantics of the different models with respect to the stated requirements. *In theory*, ISO/IEC 9126 seems well suited for Software Quality Engineering. Further research is needed to see if the measures associated with ISO/IEC 9126 make this model usable for Software Quality Engineering *in practice*.

References

1 Adey, C. A. & Hill, G. K. (2000). *Quality / ISO 9000 as a Marketing Tool*, [En ligne]. http://www.smps.org/mrc/articles/0200qualityiso.pdf

2 Bazzana, G., Anderson, O., & Jokela, T. (1993). *ISO 9126 and ISO 9000: Friends or foes?* Presented at Software Engineering Standards Symposium.

3 Biehl, R. E. (2001). *Six sigma for Software.* IEEE Software, 21(2), 68-70.

4 Boddie, J. (2000). *Do We Ever Really Scale Down?*, IEEE Software,17(5), 79-81.

5 Boehm, B. W., Brown, J. R., Kaspar, J. R., Lipow, M. L. & MacCleod, G. (1978). *Characteristics of Software Quality.* New York: American Elsevier.

6 Boehm, B. W., Brown, J. R., Lipow, M. L. (1976). *Quantitative Evaluation of Software Quality.* Proceedings of the 2nd international conference on Software engineering, San Fransisco, California, United States, 592-605, IEEE Computer Society Press.

7 Bourque, P., R. Dupuis, A. Abran, J.W. Moore, L.L. Tripp et S. Wolff, (2000) *Fundamental Principles of Software Engineering - A Journey*, Journal of Systems and Software,

8 Côté, M.-A., Suryn, W., Martin, R. A., Laporte, C. Y. (2004a). *Evolving a Corporate Software Quality Assessment Exercice: A Migration Path to ISO/IEC 9126*, Software Quality Professional, 6(3), 4-17.

9 Côté, M.-A., Suryn, W., Martin, R. A., Laporte, C. Y. (2004b). *The analysis of the industrial applicability of software product quality ISO standards: the context of MITRE's Software Quality Assessment exercise*, in Proceedings of the 12th International Software Quality Management & INSPIRE Conference (BSI) 2004, Canterbury, Kent, United Kingdom.
10 Côté, M.-A., Suryn, W., Laporte, C. Y., Martin, R. A. (2005). *The Evolution Path for Industrial Software Quality Evaluation Methods Applying ISO/IEC 9126:2001 Quality Model: Example of MITRE's SQAE Method*, Software Quality Journal, vol. 13, 17-30.
11 Crosby, P.B. (1979). *Quality is free: The art of making quality certain.* New York : McGraw-Hill.
12 Diaz M. & Sligo, J. (1997). *How Software Process Improvement Helped Motorola*, IEEE Software, 17(5), 75-81.
13 Dromey, R. G. (1995). *A model for software product quality.* IEEE Transactions on Software Engineering 21, 146-162.
14 Dromey, R. G. (1996). *Cornering the Chimera.* IEEE Software, 13(1), 33-43.
15 Eickelman, N. (2003). *An Insider's View of CMM Level 5*, IEEE Software, 20(4), 79-81.
16 Haley, T. J. (1996). *Software Process Improvement at Raytheon*, IEEE Software, 13(6), 33-41.
17 *Georgiadou E. (2003 a)* Software Process and Product Improvement, A Historical Perspective, *International Journal of Cybernetics, Volume 1, No1, Jan 2003 pp172-197*
18 Georgiadou E.(2003b) *GEQUAMO– A Generic, Multilayered, Customisable, Software Quality Model*, Volume 11, Number 4 , 313-323 November 2003
19 Highsmith, J. (2002). *Agile Software Development Ecosystems*, Addison-Wesley Professional.
20 IEEE. 1998. *Std. 1061-1998 IEEE Standard for a Software Quality Metrics Methodology.*
21 ISO/IEC. 1999a. *ISO/IEC 14598-1: Software product evaluation-Part 1 : General overview*. Geneva, Switzerland: International Organization for Standardization.
22 ISO/IEC. 1999b. *ISO/IEC 9000:2000 Quality management systems -- Fundamentals and vocabulary* . Geneva, Switzerland: International Organization for Standardization.
23 ISO/IEC. 2000. *ISO/IEC 15288: System Life Cycle Processes.* Geneva, Switzerland: International Organization for Standardization.
24 ISO/IEC. 2001a. *ISO/IEC 9126-1: Software Engineering-Software product quality-Part 1 : Quality model.* Geneva, Switzerland: International Organization for Standardization.
25 ISO/IEC. 2001b. *ISO/IEC DTR 9126-4: Software engineering-Software product quality-Part 4: Quality in use metrics.* Geneva, Switzerland: International Organization for Standardization.
26 ISO/IEC. 2003a. *ISO/IEC TR 9126-2: Software Engineering-Software product quality-Part 2 : External metrics.* Geneva, Switzerland: International Organization for Standardization.

27 ISO/IEC. 2003b. *ISO/IEC TR 9126-3: Software engineering-Software product quality-Part 3: Internal metrics*. Geneva, Switzerland: International Organization for Standardization.
28 Kitchenham, S. L., Pfleeger (1996). *Software Quality: The Elusive Target*. IEEE Software,13(1), 12-21.
29 Laitinen, M. (2000). *Scaling Down is Hard to Do*, IEEE Software, 17(5), 78-80.
30 Leffingwell, D. & Widrig, D. (1999). *Managing Software Requirements, A Unified Approach*. Addison-Wesley Professional.
31 Martin, R. A. & Shaffer, L. (1996). *Providing a framework for effective software quality assessment*. Bedford, Mass : MITRE Corporation.
32 McCall, J. A., Richards, P. K., & Walters, G. F. (1977). *Factors in software quality*. Griffiths Air Force Base, N.Y. : Rome Air Development Center Air Force Systems Command.
33 Pfleeger, S. L. (2001). *Software Engineering: Theory and practice* (2nd ed.). Upper Saddle River, N.J. : Prentice Hall.
34 Pressman, R. S. (2001). *Software Engineering: A practitioner's approach* (5th ed.). Boston: McGraw-hill.
35 Siaka K V., Berki E, Georgiadou E, Sadler C (1997): The Complete Alphabet of Quality Software Systems: Conflicts and Compromises, 7th World Congress on Total Quality&Qualex 97, New Delhi, India, 17-19 February
36 Siaka, K.V., Georgiadou, E. PERFUMES: A Scent of Product Quality Characteristics, SQM 2005, March 2005, UK
37 Suryn, W. (2003). *Course notes SYS861*. École de Technologie Supérieure, Montréal.
38 Voas, J. (2003). *Assuring Software Quality Assurance*. IEEE Software, 20(3), 48-49.