Software Quality: The Elusive Target

Deepak Dagar*
Dr. Amit Gupta*

Abstract

In the recent past, when bank statements contained errors or the telephone network broke down, the general public usually blamed "the computer," making no distinction between hardware and software. However, high-profile disasters are alerting more people to the crucial nature of software quality in their everyday lives. Before long, we can expect increasing public concern about the pervasiveness of software, not only in public services but also in consumer products like automobiles, washing machines, telephones, and electric shavers. Consequently, we software professionals need to worry about the quality of all our products from large, complex, stand-alone systems to small embedded ones.

So how do we assess "adequate" quality in a software product? The context is important. Errors tolerated in word-processing software may not be acceptable in control software for a nuclear-power plant. Thus, we must reexamine the meanings of "safety-critical" and "mission-critical" in the context of software's contribution to the larger functionality and quality of products and businesses. At the same time, we must ask ourselves who is responsible for setting quality goals and making sure they are achieved.

1. Introduction

1.1 What Does Quality Really Mean?

Measurements let us know if our techniques really improve our software, as well as how process quality affects product quality. We also need to know how the quality we build in can affect the product's use after delivery and if the investment of time and resources to assure high quality reap higher profits or larger market share. In other words, we want to know if good software is good business. Still, most people believe that quality is important and that it can be improved. Companies and countries continue to invest a great deal of time, money, and effort in improving software quality.

But we should try to determine if these national initiatives have directly affected and improved software quality. The answer may depend on how you approach quality improvement. Some companies take a product-based approach, while others focus on process.

1.2 Views of Software Quality

Quality is a complex and multifaceted concept" that can be described from five different perspectives.

- The Transcendental view sees quality as something that can be recognized but not defined.

*Department of Computer Science, Maharaja Agrasen Institute of Management Studies, Delhi, India

- The user view sees quality as fitness for purpose.
- The manufacturing view sees quality as conformance to specification.
- The product view sees quality as tied to inherent characteristics of the product.
- The value-based view sees quality as dependent on the amount a customer is willing to pay for it.

1.2.1 Transcendental view
This view of software quality is much like Plato's description of the ideal or Aristotle's concept of form. Just as every table is different but each is an approximation of an ideal table, we can think of software quality as something toward which we strive as an ideal, but may never implement completely. When software gurus exhort us to produce products that delight users, this delight represents the strived-for "recognition" in the transcendental definition of quality.

1.2.2 User view
Whereas the transcendental view is ethereal, the user view is more concrete, grounded in product characteristics that meet the user's needs. This view of quality evaluates the product in a task context and can thus be a highly personalized view. In reliability and performance modeling, the user view is inherent, since both methods assess product behavior with respect to operational profiles (that is, to expected functionality and usage patterns). Product usability is also related to the user view: in usability laboratories, researchers observe how users interact with software products.

1.2.3 Manufacturing view
The Manufacturing view focuses on product quality during production and after delivery. This view examines whether or not the product was constructed "right the first time," in an effort to avoid the costs associated with rework during development and after delivery. This process focus can lead to quality assessment that is virtually independent of the product itself. That is, the manufacturing approach adopted by ISO 9001 [1] and the Capability Maturity Model[2] advocates conformance to process rather than to specification.

There is little evidence that conformance to process standards guarantees good products. Although process standards are usually based on the principle of "documenting what you do and doing what you say," both CMM and ISO 9001 also insist (with different degrees of emphasis) that you improve your process to enhance product quality [3].

1.2.4 Product view
Whereas the user and manufacturing views examine the product from without, a product view of quality looks inside, considering the product's inherent characteristics.

This approach is frequently adopted by software-metrics advocates, who assume that measuring and controlling internal product properties (internal quality indicators) will result in improved external product behavior (quality in use). Assessing quality by measuring internal properties is attractive because it offers an objective and context-independent view of quality.

However, more research is needed to confirm that internal quality assures external quality and to determine which aspects of internal quality affect the product's use.

1.2.4 Value Based view
Different views can be held by different groups involved in software development. Customers or marketing groups typically have a user view, researchers a product view, and the production department a manufacturing view. If the difference in viewpoints is not made explicit, misunderstandings about quality created during project initiation are likely to resurface as (potentially) major problems during product acceptance.

These disparate views can complement each other in early phases. If the user's view is stated explicitly during requirements specification, the technical specification that drives the production process can be derived directly from it as can product functionality and features. However, problems can arise when changes to the requirements occur. At this point, the user's requirement for a useful product may be in conflict with the manufacturer's goal of minimizing rework.

This is where the value-based view of quality becomes important. Equating quality to what the customer is willing to pay for encourages everyone to consider the trade-offs between cost and quality. A value-based perception can involve techniques to manage conflicts when requirements change. Among them are "design to cost," in which design possibilities are constrained by available resources and "requirements scrubbing," in which requirements are assessed and revised in light of costs and benefits.

1.3 Measuring Quality
The perspective we take on quality influences how we define it. But we also want to be able to measure quality so we can establish baselines, predict likely quality, and monitor improvement. Here, too, perspective influences our choice. Users access software-product quality in terms of their interaction with the final product. Product attributes that contribute to user satisfaction are a mixture of:

- The product's functions, which are either present or absent;
- The product's nonfunctional qualities (its behavior), which is measurable within some range;
- The constraints that determine if a customer will use a particular product.

For example, a system may be required to perform a particular function, and a nonfunctional requirement may prescribe that the function be performed within two seconds of its invocation. At the same time, the system is constrained by the function's cost and availability, as well as the environment it will be used in. Past discussions of product quality have ignored constraints, which are considered to be the responsibility of managers who consider trade-offs between quality and cost. Some quality experts now suggest that all aspects of quality related to user needs be considered during definition and assessment. This corresponds to the ISO definition of quality, "the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs [4]".

1.3.1 Measuring the user's view
When users think of software quality, they often think of reliability: how long the product functions properly between failures. Reliability models plot the number of failures over time [5]. These models sometimes use an operational profile, which depicts the likely use of different system functions [6].

Users, however, often measure more than reliability. They are also concerned about usability, including ease of installation, learning, and use. These characteristics can be measured directly [7]. For example; learning time can be captured as the average elapsed time (in hours) for a typical user to achieve a stated level of competence. The quality concept is broken down into component parts until each can be stated in terms of directly measurable attributes. Thus, each quality-requirement specification includes a measurement concept, unit, and tool, as well as the planned level (the target for good quality), the currently available level, the best possible level (state-of-the-art), and worst level.

1.3.1 Measuring the manufacturer's view

The manufacturing view of quality suggests two characteristics to measure: defect counts and rework costs.

a. Defect counts

Defect counts are the number of known defects recorded against a product during development and use. For comparison across modules, products, or projects, you must count defects in the same way and at the same time during the development and maintenance processes. For more detailed analysis, you can categorize defects on the basis of the phase or activity where the defect was introduced, as well as the phase or activity in which it was detected. This information can be especially helpful in evaluating the effects of process change (such as the introduction of inspections, tools, or languages).

To compare the quality of different products, you can "normalize" defect count by product size, to yield a defect density. This measure lets you better compare modules or products that differ greatly in size. In addition, you can "normalize" post release defect counts by the number of product users, the number of installations, or the amount of use. Dividing the number of defects found during a particular development stage by the total number of defects found during the product's life helps determine the effectiveness of different testing activities.

b. Rework Cost

Defects differ in their effect on the system: some take a little time to find and fix; others are catastrophic and consume valuable resources. To monitor the effect of defect detection and correction, we often measure rework costs the staff effort spent correcting defects before and after release. This cost of nonconformance supports the manufacturing view. Rework is defined as any additional effort required to find and fix problems after documents and code are formally signed-off as part of configuration management. Thus, end-phase verification and validation are usually excluded, but debugging effort during integration and system testing is included. To compare different products, rework effort is sometimes "normalized" by being calculated as a percentage of development effort.

Because we want to capture the cost of nonconformance, we must be sure to distinguish effort spent on enhancements from effort spent on maintenance. Only defect correction should count as rework. It is also important to separate pre- and post-release rework. Post-release rework effort is a measure of delivered quality; prerelease rework effort is a measure of manufacturing efficiency. If we can attribute the prerelease rework effort to specific phases, we can use it to identify areas for process improvement.

Developers and customers alike are interested in knowing as early as possible the likely quality of the delivered product. But the relationship between post-delivery failure and defects, structural measures, and other pre-delivery information is far from clear. In 1984, the Esprit-funded Request project concluded that there were no software-product metrics that were likely to be good predictors of final product qualities [8]. Much useful software-metrics research concentrates instead on linking software- product measures to error-prone modules.

1.4 Capturing Quality Data

The way we measure quality depends on the viewpoint we take and the aspect of quality we want to capture. Peter Mellor provides guidelines for defining incidents, failures, and faults that can help you capture raw data for reliability assessment [9]. This type of data can measure other aspects related to the user view of quality. Proper classification of incidents lets us identify potential usability problems (that is, incidents resulting from misuse of the software or misunderstanding of the user manuals and help systems). In addition, information about the time and effort needed to diagnose the cause of different priorities and correct any underlying faults can give us useful information about system maintainability. This sort of data is often used to monitor service-level agreements that define the obligations of software-maintenance organizations.

Capturing data associated with other quality aspects particularly those associated with the product and manufacturing view is usually part of a company's software- measurement system. The particular measures an organization collects will depend on its goals and management requirements. Techniques such as the Goal-Question-Metric paradigm developed by Vic Basili and colleagues [10] can help us identify which measures will help us monitor and improve quality.

1.5 Modeling Quality

In the past, many researchers developed software quality models that were intended to be comprehensive and applicable to all software development.

1.5.1 McCall's Quality Model

One of the earliest quality models was suggested by Jim McCall

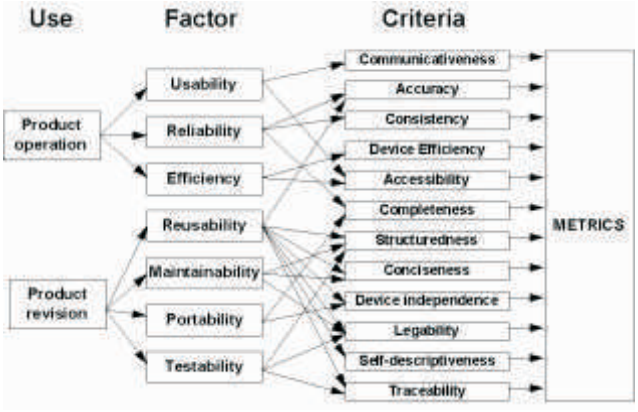

Figure 1: McCall's quality model defines software-product qualities as a hierarchy of factors, criteria, and metrics.

[11]. As shown in Figure 1, the model defines software-product qualities as a hierarchy of factors, criteria, and metrics. The arrows indicate which factors the criteria influence.

A quality factor represents a behavioral characteristic of the system. A quality criterion is an attribute of a quality factor that is related to software production and design. A quality metric is a measure that captures some aspect of a quality criterion. Thus, the 11 quality factors contribute to a complete picture of software quality.

One or more quality metric should be associated with each criterion. Thus, as the figure shows, you can measure portability by combining self-descriptiveness, modularity, software-system independence, and machine independence. The metrics are derived from the number of "yes" responses to questions whose answers are subjective, such as "Is all documentation structured and written clearly and simply such that procedures, functions, algorithms, and so forth can easily be understood?"

Dividing the number of yes responses by the number of questions gives a series of values in the range 0 to 1. However, there are problems with values derived in this way. The degree of subjectivity varies substantially from one question to another, even though all responses are treated equally. This variation makes combining metrics difficult, if not impossible.

1.5.2 ISO 9126

More recently, international efforts have led to the development of a standard for software-quality measurement, ISO 9126[12] .The standards group has recommended six characteristics to form a basic set of independent quality characteristics. The quality characteristics and their definitions are shown in Table 1.

Table 1: ISO 9126 Quality Characteristic

Quality Characteristic	Definition
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 sub-characteristics, as Figure 2 shows. The arrows in the figure indicate how the characteristic are decomposed into sub-characteristics.

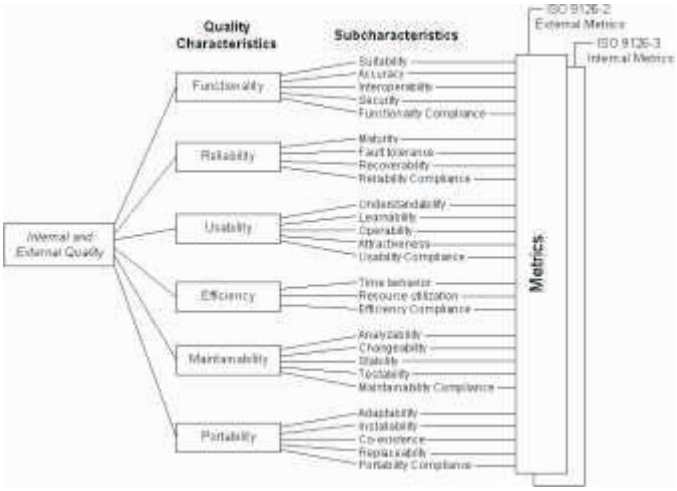

Figure 2: The ISO 9126 sample quality model refines the standard's features into sub-characteristics, as the arrows indicate.

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.

Although the ISO 9126 model is similar to McCall's, there are several differences.

Clearly, the ISO model uses a different quality framework and terminology, 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 sub-characteristics to refine the characteristic
- Indicators to measure quality sub-characteristics
- 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. The ISO framework is completely hierarchical each sub-characteristic is related to only one characteristic. Also, the sub-characteristics 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.

1.6 Models Problems

The two models presented here are representative of older quality models. Although their approaches differ, the models share common problems.

First, they lack a rationale for determining which factors should be included in the quality definition. They also lack a rationale for deciding which criteria relate to a particular factor. Thus, the selection of quality characteristics and sub-characteristics can seem arbitrary.

For example, it is not clear why portability is a top-level characteristic of ISO 9126, but interoperability is a sub-characteristic of functionality. This lack of rationale makes it impossible to determine if the model is a complete or consistent definition of quality.

Second, there is no description of how the lowest-level metrics (called indicators in the ISO 9126 model) are composed into an overall assessment of higher level quality.

1.6 The Business Value of Quality

In the last few decades, software has grown to become a vital part of most companies' products and services. With that growth comes our responsibility for determining how much software contributes to the corporate bottom line. When a telephone company cannot implement a new service because the billing-system software cannot handle the new features, then lack of software quality is a corporate problem.

And when software problems stop the assembly line, ground the plane or send the troops to the wrong location, organizations realize that software is essential to the health and safety of business and people.

In particular, we must look more carefully at how our methods and tools affect software quality. Businesses take big risks when they invest in technology that has not been carefully tested and evaluated.

But looking at the software alone is not enough. We must see it in the context of how it is used by business to determine if investment in higher software quality is worthwhile. As Ed Yourdon pointed out, sometimes less-than-perfect is good enough;[13] only business goals and priorities can determine how much "less than perfect" we are willing to accept.

Conclusion

Quality is a complex concept. Because it means different things to different people, it is highly context-dependent. Just as there is no one automobile to satisfy everyone's needs, so too there is no universal definition of quality. Thus, there can be no single, simple measure of software quality acceptable to everyone. To assess or improve software quality in your organization, you must define the aspects of quality in which you are interested, and then decide how you are going to measure them.

By defining quality in a measurable way, you make it easier for other people to understand your viewpoint and relate your notions to their own. Ultimately, your notion of quality must be related to your business goals. Only you can determine if good software is good business.

References

1. *ISO 9001 Quality Systems - Model for Quality Assurance in Design/Development, Production, Installation, and Servicing, International Organization for Standardization, Geneva, 1994.*
2. *M. Paulk et al., "Capability Maturity Model, Version 1.1," IEEE Software, July 1993, pp. 18-27.*
3. *M. Paulk, "How ISO 9001 Compares With the CMM," IEEE Software, Jan. 1995, pp. 74-83.*
4. *ISO 8402 Quality Management and Quality Assurance - Vocabulary, International Organisation for Standardization, Geneva, 2nd Edition, 1994.*
5. *Special issue on reliability measurement, IEEE Software, July 1992.*
6. *J. Musa, "Operational Profiles in Software-Reliability Engineering," IEEE Software, March 1993, pp. 14-32.*
7. *T. Gilb, Principals of Software Engineering Management, Addison-Wesley, Reading, Mass., 1987.*
8. *G. Frewin et al., "Quality Measurement and Modeling State of the Art Report," Request Report to the CEC Esprit program, R1.1.1, 1984 (available from the European Commission, Brussels).*
9. *J.A. McCall, P.K. Richards, and G.F. Walters, Factors in Software Quality, Vol. 1, 2, and 3, AD/A-049-014/015/055, Nat'l Tech. Information Service, Springfield, Va., 1977.*
10. *ISO9126 Information Technology -Software Product Evaluation Quality-Characteristics and Guidelines for Their Use, International Organization for Standardization, Geneva, 1992.*
11. *P. Mellor, "Failures, Faults and Changes in Dependability Measurement," J. Information and Software Technology, Oct. 1992, pp. 640-654.*
12. *V. Basili and D. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments," IEEE Trans. Software Eng., June 1989, pp. 758-773.*
13. *E. Yourdon, "When Good Enough Software is Best," IEEE Software, May 1995, pp. 79-81.*