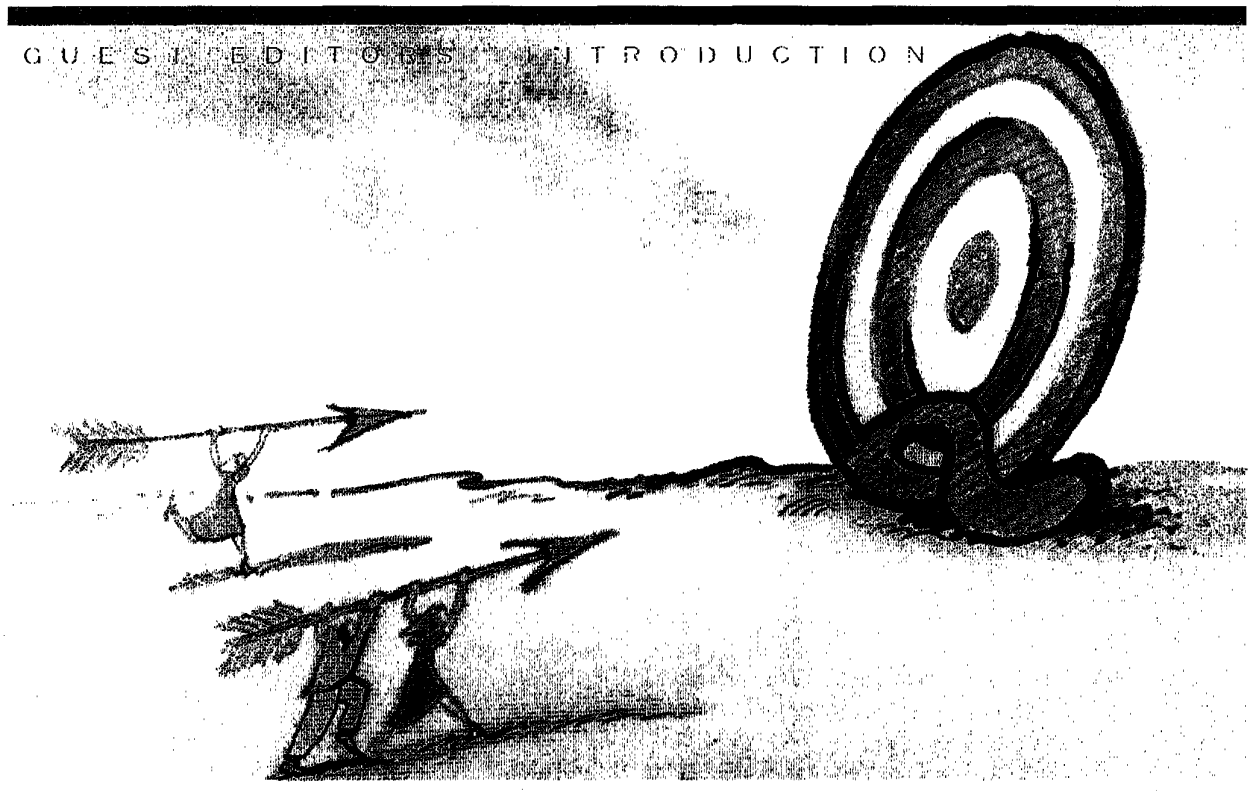

SOFTWARE QUALITY: THE ELUSIVE TARGET

BARBARA KITCHENHAM, *National Computing Centre*
SHARI LAWRENCE PFLEEGER, *Systems/Software, Inc.*

If you are a software developer, manager, or maintainer, quality is often on your mind. But what do you really mean by software quality? Is your definition adequate? Is the software you produce better or worse than you would like it to be? In this special issue, we put software quality on trial, examining both the definition and evaluation of our software products and processes.

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 and the ensuing debates in the press 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.

WHAT DOES QUALITY REALLY MEAN?

Most of us are affected by the quality of the software we create because our organization's viability depends on it. And most software-related tools and methods — including those described in *IEEE Software* — claim to assess or im-

0740-7459/96/$05.00 © 1996 IEEE

prove software quality in some way. So we must question what we and our customers mean by software quality.

A good definition must let us measure quality in a meaningful way. 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.

Recent articles have raised this question, but the answer is still far from clear. 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; both strategies have led to Malcolm Baldridge awards for overall product quality.

In their more general questioning of quality goals and techniques, Roger Howe, Dee Gaeddert, and Maynard Howe pointed out that most quality initiatives either fail (by drowning in a sea of rhetoric) or cannot demonstrate success because no financial return can be identified.[1] In this special issue, we question software quality in the same way. We consider the meaning of software quality, how we assess it, and whether the steps we are taking to improve it are really worthwhile.

VIEWS OF SOFTWARE QUALITY

In an influential paper examining views of quality, David Garvin studied how quality is perceived in various domains, including philosophy, economics, marketing, and operations management.[2] He concluded that "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.
- 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.

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.

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 in-

SOFTWARE QUALITY SURVEY

Last year we invited readers to air their views on software quality by completing a short questionnaire. As promised, we report the results of the survey here. We thank those who completed the questionnaire. Of the 27 respondents, 17 were from the US, five from Europe, and five from Asia — similar to the distribution of *IEEE Software* readers. Respondents background experience was mixed and some individuals marked several categories: there were nine marks in development and seven in research. Of the 12 who marked "other," five wrote in "quality assurance."

Although this is neither a large nor representative sample, we hope that the responses will make you think about your own perspective on quality and examine how quality is effected by the best-practice activities you implement.

Views of quality. We asked respondents to suggest their own quality definitions and then assessed them against David Garvins' five perspectives. Most definitions emphasized the user or manufacturing view (17 and 13, respectively). However, 14 respondents suggested definitions that covered two or more different views. Other viewpoints included product (nine respondents), transcendental (three), and value (five).

Of the 27 respondents, 18 strongly agreed and seven agreed that software quality constituted a problem. When asked if a quality-management system alone could solve the quality problem, 18 disagreed, three agreed, and two respondents strongly agreed.

Opinions were mixed on whether quality is more or less important than time to market: 17 thought time to market was of equal or greater importance than quality and seven thought quality was more important (three had no opinion). On the relative value of quality and productivity, 17 thought quality was more important than productivity and nine thought productivity was of equal or greater importance.

Quality issues. We asked respondents to select three quality issues from a list and rank them in terms of importance. We used a simple ranking order for the responses, assigning three points to the items marked as most important, two points to those marked as next most important, and one point to the third most important issue.

Respondents ranked specifying quality requirements objectively as most important (28 points), followed by setting up a quality-management system (20 points). This implies that respondents rate a QMS as necessary — but not sufficient — for addressing quality. Achieving operational quality that meets requirements was ranked third (18 points), followed by measuring quality achievements (17 points), and agreeing with the customer on what quality means (15 points).

herent, 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.

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[3] and the Capability Maturity Model[4] — advocates conformance to process rather than to specification.

There is little evidence that conformance to process standards guarantees good products. In fact, critics of this view suggest that process standards guarantee only uniformity of output and can thus institutionalize the production of mediocre or bad products. However, this criticism may be unfair. 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.[5]

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. Some researchers have developed models to link the product view to the user view.

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.

Product purchasers take a rather different value-based view. Part of their job is to know if a software product represents value for money to their organization. In this context, internal software measures are irrelevant. Purchasers compare the product cost with the potential benefits. In her article in this issue, "Quality Outcomes: Determining Business Value," Pamela Simmons discusses benefits obtained from investment in information systems.

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 assess 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; and
- the constraints that determine if a customer will use a particular product.

There is little evidence that conformance to process standards guarantees good products.

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."[6]

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.[7] These

models sometimes use an operational profile, which depicts the likely use of different system functions.[8]

Users, however, often measure more than reliability. They are also concerned about usability, including ease of installation, learning, and use. Tom Gilb suggests that these characteristics can be measured directly.[9] 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.

Gilb's technique can be generalized to any quality feature. 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. Gilb does not prescribe a universal set of quality concepts and measurements, because different systems will require different qualities and different measurements.

Measuring the manufacturer's view. The manufacturing view of quality suggests two characteristics to measure: defect counts and rework costs.

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).

The relationship between defects counts and operational failures is unclear. However, you can use defect counts to indicate test efficiency and identify process-improvement areas. In addition, a stable environment can help you estimate post-release defect counts.

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" postrelease 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.

Rework costs. 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 postrelease rework. Postrelease 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 predelivery 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.[10] Twelve years later, there is no evidence of any significant improvement. Much useful software-metrics research concentrates instead on linking software-product measures to error-prone modules.

An example of this type of work is found in the article by Taghi Koshgoftaar and colleagues, "Early Quality Prediction: A Case Study in Telecommunications," in this issue. However, researchers have provided no convincing evidence that module-level measures are consistently related to external, behavioral properties of the product as a whole. Indeed, there are major technical difficulties in assessing such relationships; correlating system-level measures with simple module-level statistics, such as means, are unlikely to be appropriate.

The way we measure quality depends on the viewpoint we take.

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.[11] 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.

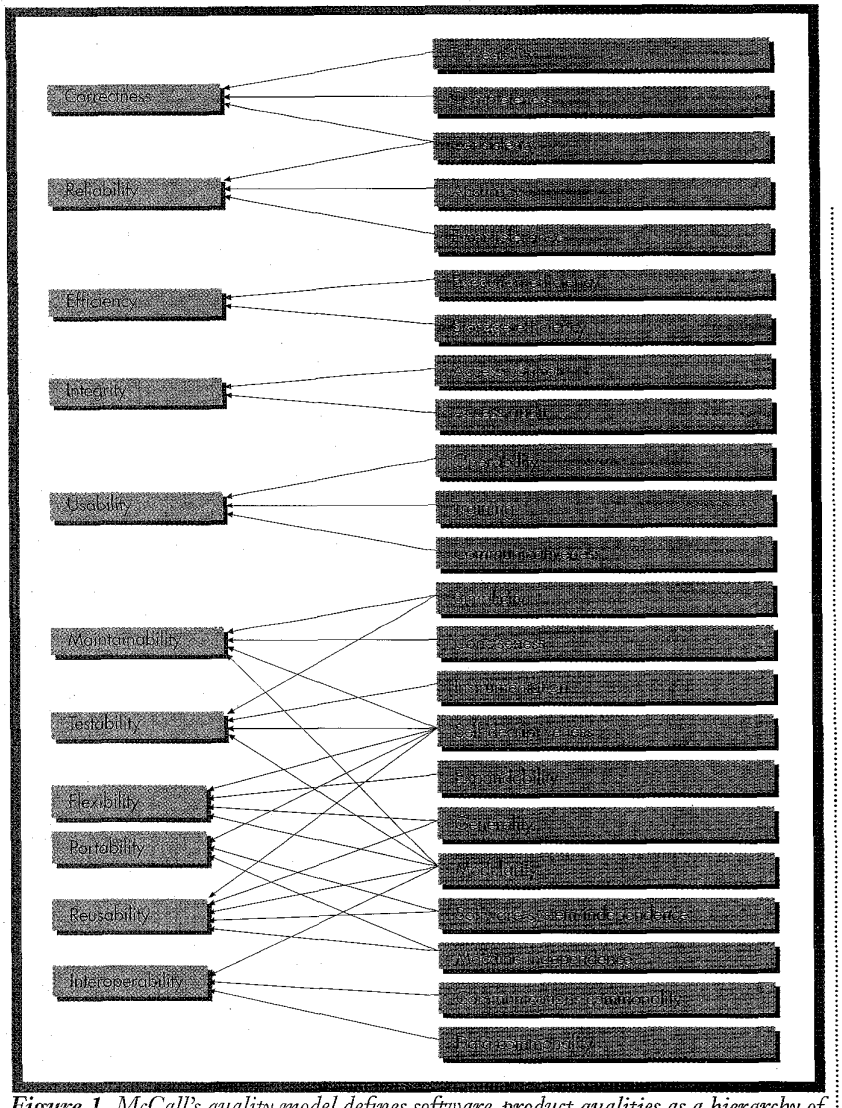

Figure 1. McCall's quality model defines software-product qualities as a hierarchy of factors, criteria, and metrics.

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[12] can help us identify which measures will help us monitor and improve quality.

MODELING QUALITY

To understand and measure quality, researchers have often built models of how quality characteristics relate to one another. Just as Gilb decomposed quality into various factors, so have others depicted quality in a hierarchical way. In the past, many researchers developed software quality models that were intended to be comprehensive and applicable to all software development.

McCall's quality model. One of the earliest quality models was suggested by Jim McCall and colleagues.[13] 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. The measures can be composed into either measures of specific factor quality or the product's quality as a whole by considering the relevant selection of questions.

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. Moreover, when appropriate, response complexity should be reflected in a richer measurement scale. For example, while it is reasonable to expect a yes-or-no response to the question, "Does this module have a single entry and exit point?" questions about documentation clarity probably require a multiple-point ordinal scale to reflect the variety of possible answers.

ISO 9126. More recently, international efforts have led to the development of a standard for software-quality measurement, ISO 9126.[14] 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.

The standard also includes a sample quality model that refines the features of ISO 9126 into several subcharacteristics, as Figure 2 shows. The arrows in the figure indicate how the characteristics are decomposed into 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 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 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.

Model 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 subcharacteristics can seem arbitrary. For example, it is not clear why portability is a top-level characteristic of ISO 9126, but interoperability is a subcharacteristic of functionality. This lack of rationale makes

TABLE 1 ISO 9126 QUALITY CHARACTERISTICS

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).

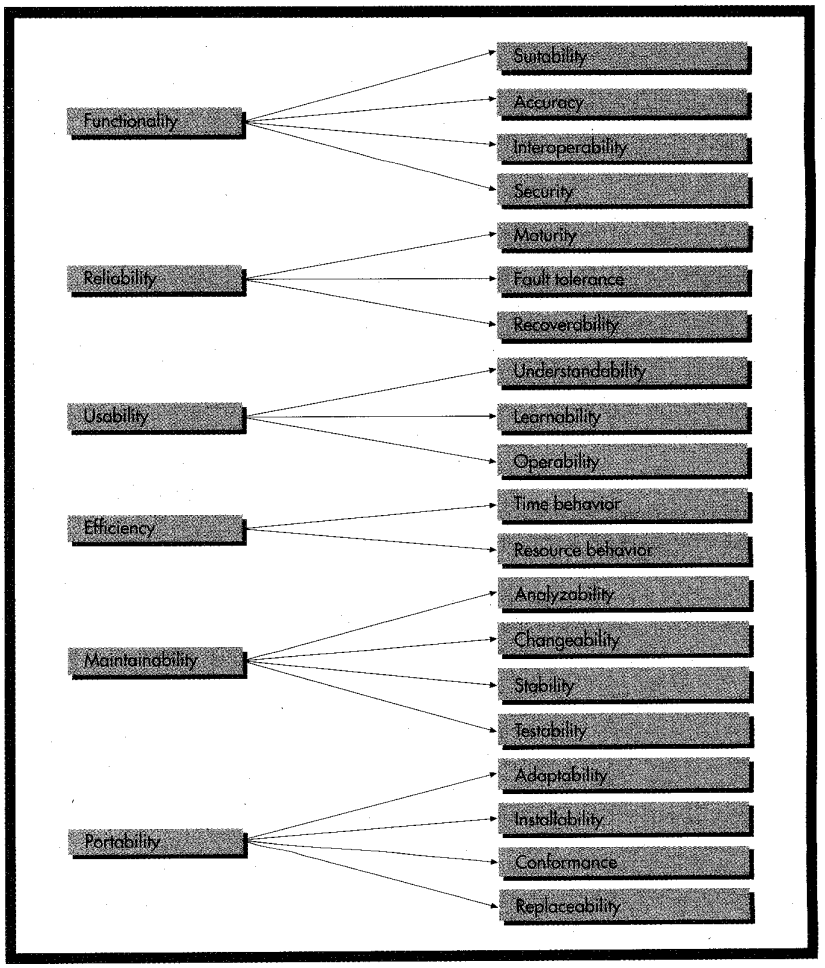

Figure 2. The ISO 9126 sample quality model refines the standard's features into subcharacterisitics, as the arrows indicate.

SIGNPOSTS AND LANDMARKS: A SOFTWARE QUALITY READING LIST

There is a lot of literature on software quality, as well as conferences and workshops that address its key components. Here we highlight a few of the significant papers and meetings.

David Garvin's article, "What Does Product Quality Really Mean?" (*Sloan Management Review*, Fall 1984), asks some probing questions about what we really mean by quality. He suggests that quality is in the eye of the beholder and that the importance and meaning of quality differ according to your perspective. These perspectives can be useful in deciding what quality means to you and in developing quality measures to help improve your processes and products.

Quality models. One of the first software quality models to be developed and publicized widely was by Jim McCall and his colleagues from the US Air Force Rome Air Development Center (now Rome Laboratories). The report, "Factors in Software Quality" (Tech. Report AD/A-049-014/015/055, NTIS, Springfield, Va., 1977), presents quality as a hierarchy of factors, each of which is associated with a metric. Although it is quoted widely and often reproduced in textbooks and papers, it is rarely used in practice.

Recently, ISO Technical Committee JTC1/SC7/WG6 produced a standard for measuring software quality, *ISO 9126, Information Technology: Software Product Evaluation — Quality Characteristics and Guidelines for Their Use*. ISO 9126 is a hierarchical model and claims to be complete and comprehensive. However, the standard is currently being reexamined, and we anticipate that attempts to use 9126 in its current form on real projects will result in substantial revision to the standard.

Geoff Dromey's paper, "A Model for Software-Product Quality" (*IEEE Trans. Software Eng.*, Feb. 1995), addresses some of the key problems with current software quality models. He presents a framework for developing models that includes criteria for accepting quality factors and for knowing when the model is complete.

Process or product? In addition to deciding what quality means, we have to decide whether to improve it by using the process or monitoring the product. "Making War on Defects," a set of articles in *IEEE Spectrum* (Sept. 1993), describes two very different companies, AT&T and Motorola, and discusses how their approaches to quality improvement led to award-winning product quality. AT&T focuses on product improvement, while Motorola used process improvement to accomplish the same goal.

Nancy Leveson and Clark Turner give a good example of how to analyze software quality in an article on the Therac-25 accidents in *Computer* (July, 1993). They explain software's role in the product, explore the meaning of quality, and track a user's problem to its source in inappropriate software design.

Does quality matter? Erik Brynjolfsson was the first to question whether our efforts in improving quality are really effective. In his controversial article, "The Productivity Paradox of Information Technology" (*Communications of the ACM*, Dec. 1993), Brynjolfsson presents data to show that business productivity can go down with the introduction of information technology. His focus on how information technology affects business goals has created a stir, and subsequent articles (even by him) have refuted some of his findings.

In the book *Quality on Trial* (McGraw Hill, 1992), Roger Howe, Dee Gaeddert, and Maynard Howe also question whether an emphasis on quality delivers benefits to a company and its customers. Their conclusion is that most quality-management systems are too inward-looking to deliver benefits to customers. They point out that improving internal administrative and management systems does not readily equate to better, more competitive products.

Other sources. You will find many articles on software quality issues in *IEEE Software*, *IEEE Transactions on Software Engineering*, *IEEE Transactions on Reliability*, and *Software Quality Journal*. There are also many conferences devoted to software quality: The Pacific Northwest Quality Conference held annually in Portland, Oregon; Quality Week, held in San Francisco every May (sponsored by *Software Magazine*); and The European Software Quality Conference. The American Society for Quality Control often sponsors other conferences on software quality.

Other standards related to software quality are:

- *ISO 9001 Quality Systems — Model for Quality Assurance in Design/Development, Production, Installation, and Servicing*, International Organization for Standardization, Geneva, 1994.
- *ISO 8402 Quality Management and Quality Assurance — Vocabulary*, International Organisation for Standardization, Geneva, 2nd Edition, 1994.

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 characteristics. In particular, then, there is no means for verifying that the chosen metrics affect the observed behavior of a factor. That is, there is no attempt to measure factors at the top of the hierarchy, so the model is untestable.

Dromey's model. Geoff Dromey has developed a model that addresses many of these problems. Dromey points out that hierarchical models that use a top-down decomposition are usually rather vague in their definitions of lower levels. They thus offer little help to software developers who need to build quality products. Dromey believes that it is impossible to build high-level quality attributes such as reliability or maintainability into products. Rather, software engineers must build components that exhibit a consistent, harmonious, and complete set of product properties that result in the manifestations of quality attributes. His article, "Cornering the Chimera," in this issue, describes his approach in more detail.

Dromey's approach is important because it allows us to verify models. It establishes a criterion for including a particular software property in a model (that is, that a quality defect can be associated with the concept) and a means of establishing when the model is incomplete (that the model cannot classify a specific defect).

Modeling process quality. Another approach to quality modeling is to look at process. John Evans and John Marciniak suggested a full process model analogous to the product quality models described above.[15] The quality implications of a specific software-development process are also of interest. For example, in "Support for Quality-Based Design and Inspection" in this issue, Ilkka Tervonen discusses how to integrate the inspection process with designing for quality by justifying different development decisions based on their impact on quality requirements.

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. When a national gas utility must spend millions of dollars to fix a software glitch in monitoring systems embedded in gas meters throughout the country, then small software defects become big headaches. 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. Little research is done into the relationship between software quality and business effectiveness and efficiency. But unless we begin to look at these issues, companies will be unable to support key business decisions.

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. The Desmet project, funded by the UK's Department of Trade and Industry, has produced guidelines for how to conduct case studies and experiments in support of technology evaluation.[16]

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;[17] only business goals and priorities can determine how much "less than perfect" we are willing to accept. In their article, "Software Quality in Consumer Electronic Products," Jan Rooijmans and colleagues take up this issue with a discussion of the problems and challenges associated with producing consumer electronic products.

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, 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. R. Howe, D. Gaeddert, and M. Howe, *Quality on Trial*, McGraw-Hill Europe, Maidenhead, England, 1992.
2. D. Garvin, "What Does "Product Quality" Really Mean?" *Sloan Management Review*, Fall 1984, pp. 25-45.
3. *ISO 9001 Quality Systems - Model for Quality Assurance in Design/Development, Production, Installation, and Servicing*, International Organisation for Standardization, Geneva, 1994.
4. M. Paulk, et al., "Capability Maturity Model, Version 1.1," *IEEE Software*, July 1993, pp. 18-27.
5. M. Paulk, "How ISO 9001 Compares With the CMM," *IEEE Software*, Jan. 1995, pp. 74-83.
6. *ISO 8402 Quality Management and Quality Assurance - Vocabulary*, International Organisation for Standardization, Geneva, 2nd Edition, 1994.
7. Special issue on reliability measurement, *IEEE Software*, July 1992.
8. J. Musa, "Operational Profiles in Software-Reliability Engineering," *IEEE Software*, Mar. 1993, pp. 14-32.
9. T. Gilb, *Principals of Software Engineering Management*, Addison-Wesley, Reading, Mass., 1987.
10. G. Frewin et al., "Quality Measurement and Modelling — State of the Art Report," Request Report to the CEC Esprit program, R1.1.1, 1984 (available from the European Commission, Brussels).
11. 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.
12. *ISO9126 Information Technology — Software Product Evaluation — Quality Characteristics and Guidelines for Their Use*, International Organisation for Standardization, Geneva, 1992.
13. P. Mellor, "Failures, Faults and Changes in Dependability Measurement," *J. Information and Software Technology*, Oct. 1992, pp. 640-654.
14. V. Basili and D. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments," *IEEE Trans. Software Eng.*, June 1989, pp. 758-773.
15. M. Evans and J. Marciniak, *Software Quality Assurance and Management*, John Wiley & Sons, New York, 1987.
16. B. Kitchenham, L. Pickard, and S. Lawrence Pfleeger, "Case Studies for Method and Tool Evaluation," *IEEE Software*, July 1995, pp. 52-62; correction, Sept. 1995, pp. 98-99.
17. E. Yourdon, "When Good Enough Software is Best," *IEEE Software*, May 1995, pp. 79-81.

Barbara Kitchenham is a software-engineering consultant at the National Computing Centre. Her interests are software metrics and their application to project management, quality control, and evaluation of software technologies. She was a programmer for ICL's operating system division before becoming involved with a number of UK and European research projects on software quality, software-cost estimation, and evaluation methodologies for software technology. She has written more than 30 papers on software metrics.

Kitchenham received a PhD from the University of Leeds. She is an associate fellow of the Institute of Mathematics and Its Applications and a fellow of the Royal Statistical Society.

Shari Lawrence Pfleeger is president of Systems/Software, a consultancy that specializes in software engineering and technology transfer. Her clients include major corporations, government agencies, and universities. Pfleeger has been a principal scientist at both the Contel Technology Center and Mitre, as well as a visiting professorial research fellow at the Centre for Software Reliability in London. She has written three software-engineering texts and several dozen articles.

Pfleeger received a PhD in information technology from George Mason University. She is an adviser to *IEEE Spectrum*, a member of the ACM, and associate editor-in-chief of *IEEE Software*.

Address questions about this article to Kitchenham at National Computing Centre, Oxford House, Oxford Rd., Manchester M1 7ED, UK; barbara.kitchenham@ncc.co.uk.