3

If software quality is a perception, how do we measure it?

W. M. Gentleman
National Research Council of Canada
Ottawa, Canada, K1A 0R6. Telephone: (613) 993-3857.
Fax: (613) 952-7151. Email: gentleman@iit.nrc.ca

Abstract

For over twenty years, metrics have been being invented to measure software quality. And yet quantifying quality presupposes agreement on what constitutes quality. Quality has been portrayed as an absolute quantity, subject to objective measurements. We believe this effort has been misguided. We argue that quality, like beauty, is in the eye of the beholder – that is, that quality is not absolute, but depends on the perspective taken by the evaluator. As such, any direct measure of quality must necessarily be subjective, summarizing the impressions of some particular class of people who interact with the product. Indirect measures of quality are less objective than they may appear to be – beyond the arbitrariness of the choice of measure, and any difficulty in its interpretation, there is always the tenuous link of the metric to the perception of quality by any specific group. The need for this novel point of view is especially clearly illustrated by mathematical software.

Keywords

Software quality, measurement, assessment, subjective, objective

1 INTRODUCTION

At a recent meeting, two eminent computer scientists were discussing the numerical computing environment Matlab. 'Could Matlab be space-worthy?' asked the first. 'Never mind whether you could fly it in a spaceraft', responded the second, 'would you even depend on the results of its calculations in a spaceflight?' The implied criticism was that Matlab had not been built through the process of formal specification, traceability to requirements, formal reviews, and testing against specification which is typically demanded of safety critical

R. F. Boisvert (ed.), *Quality of Numerical Software*

software. Most numerical analysts would disagree with this criticism. As a commercial product, Matlab has been implemented by knowledgeable and skilled experts, based on well studied algorithms. Beyond testing by the developer, and widespread general use, it has been intensely used by, and indeed been the subject of research by, other experts in the area.

Weaknesses have been identified, but the consensus seems to be that the base product is solid. Has the conventional process produced similar endorsement from the users of space software? To cite a few incidents: the February 1991 Patriot missile failure illustrates that users may overlook specified limitations; the May 1992 failure of the shuttle Endeavour to recover a satellite automatically illustrates that the standard development process does not protect the unwary against well known anomalies of floating point; and the June 1996 Ariane 5 failure illustrates shortcomings with blindly believing specifications.

In the past, much of the thinking about quality has been in the context of one-on-one customer/contractor relationship. Galsworthy (1912) extolled the virtues of hand crafted custom products. Today's customer often faces a very different situation: a choice between competitive off-the-shelf products. (Note that the user of numerical subroutine libraries, or even of pre-existing numerical methods, has always been in this situation.) Each of these products has its own specification, and not only are these different, not only had the individual customer no part in creating that specification, but typically the full specification is not accessible to him. None of the specifications is likely to match any individual customer's needs exactly, so eliciting detailed requirements from him in the absence of knowledge about the available products is simply an exercise in raising frustration levels. For these reasons, 'correct implementation', in the sense that the products conform to their separate specifications, is likely to be a minor issue for the customer. Field experience is likely to be of more value than formal specification and verification.

Quantifying software quality is important because, apart from aesthetic appreciation of quality products, our purpose in examining quality is to facilitate decision making. One example of these decisions is the choice of products, where several would appear to do the job. Another example is the decision of whether to accept, and to pay for, a product that claims to meet a particular need. We are often concerned with quality and price, e.g. what quality is available for a given price, or how much extra would better quality cost. Consequently another example is in deciding what investment is worth making in order to improve the quality of a given product. In many cases what we really want to do is to predict what our own level of satisfaction with the software will be, before we have had the chance to exercise the software extensively in a particular context. This comes up both when the software is unfamiliar to us, and when it is not yet complete. The classical quality assurance motivation (e.g. IEEE, 1988) of monitoring the production process is only part of the story.

An important consequence of quality is that it engenders trust. The user feels 'If the developer took proper care of the details that I recognize the need for, the prospect is good that he also took proper care of that which I would care about if I had thought of it.' The details on which quality is assessed and trust is established may not even be ones relevant to this application. 'For me, I would say software quality occurs when I perceive that the producer appears to have appropriately addressed the issues that confront me as a user. This means mostly an absence of evidence that he or she is ignorant of something that really matters to me but also positive assurances that my concerns are reflected in the system' (Johnson, 1996). Users must

make decisions about word processors and spreadsheets, for instance, and they easily identify quality (or, more significantly, lack of quality) in such products, without ever formalizing needs. In the world of mathematical software, one of the most valuable characteristics to many users is exactly when the software takes care of troublesome but rare situations, so that the user need not even be aware that such situations might exist – and certainly would not be able to enumerate and characterize them.

For numerical software, many people used to think performance (speed, intermediate store, paging behaviour, etc.) was all that mattered for quality, but as widely available machines become more capable, this view is fading as it has for other software (Carrol, 1984).

In common parlance quality is often associated with durability and lasting value. For software, since it does not wear out, this pertains to ongoing need for the software and to resilience of the software to changes in the environment.

The definition of the term *quality* is an issue. An interesting discussion of the meaning of quality can be found in Kitchenham (1986). A surprising number of people still think software quality is simply the absence of errors. Dictionary definitions are too vague to be of much help. The only relevant definition offered by the Oxford English Dictionary (Oxford, 1993), for instance, is *peculiar excellence or superiority.* Noteworthy here is that quality cannot be discussed for something in isolation: comparison is intrinsic. Many software engineering references (e.g. Gilbert, 1983; Schach, 1990; Hailstone, 1991; Tinnirello, 1995) define software quality as *correct implementation of the specification.* Such a definition can be used during product development, but it is inadequate for facilitating comparisons between products. Standards organizations have tended to refer to meeting needs or expectations, e.g. the ISO standard ISO 8492:1986 (ISO, 1986) defines quality as *the totality of features and characteristics of a product or service that bears on its ability to satisfy stated or implied needs,* adding *(Note – In a contractual environment, needs are specified, whereas in other environments, implied needs should be identified and defined.)* IEEE Std 610.12–1990 (IEEE, 1990) defines quality as *(1) The degree to which a system, component, or process meets specified requirements. (2) The degree to which a system, component, or process meets customer or user needs or expectations.* An older IEEE definition, IEEE Std P1061–1988 (IEEE, 1988) defines *Software quality is the degree to which software possesses a desired combination of attributes.*

Software quality is often defined in terms of the fitness of the product for its purpose. However different people have different purposes for the same software. A novice casual user is probably more concerned about ease of learning, and about robustness against misuse, than about efficiency. A system integrator, planning to incorporate the software in some larger system, might be more concerned about failure detection and recovery than about ease of initial installation. A third party maintenance organization is concerned with issues such as internal documentation and adequacy of scaffolding (e.g. test harnesses, test generators, and instrumentation) that go beyond issues of direct concern to the users. These show that software quality is not absolute, but is a perception depending upon for whom the quality is evaluated. Moreover, software quality is multifaceted, and the importance of the different facets changes with the context, even for the same person at different points in time.

Consider the purposes of mathematical software products, such as numerical libraries (NAG or IMSL), a numerical computation and visualization environment (MATLAB), a symbolic

mathematics system (Maple or Mathematica), or a framework for computation on specific kinds of problems (an ocean model, a bomb code). For any of these products, in addition to the support activities for the product there are people using the product for at the very least three different purposes: 1) production computation of results needed in other disciplines, 2) teaching students about the mathematics, 3) research into developing new mathematical methods. The needs of these groups are often not just different, but conflicting – what one group would regard as quality another may regard as making the product unusable. The optimizations that make the code fast enough, and the intermediate storage compact enough, for production computation to be practical may mean the code is too complex for students to learn from, and that insights possible from intermediate results and auxiliary calculations are not available. The mathematical rigour necessary to prove that new algorithms for symbolic computation have taken all possibilities into account may be so clumsy as to make the system useless for an engineer doing exploratory derivations of formulae that would only be useful to provide insight if the results are simple enough. Flexibility provided by facilitating changes to the source code may be a necessity for users trying to do computations beyond the model that a framework directly provides, but it is a nightmare for support personnel responding to problem reports, who have no easy way of recognizing whether the problem was caused by a defect in the product or by a user change breaking something. (Even restricting user extensibility to plug-in modules still leaves this problem because an inadequate API specification can lead to subtle failures of the plug-in.)

2 QUALITY ATTRIBUTES

A particularly important distinction is between what represents quality for the user and what represents quality for the supplier of a commercial product. Lists of attributes that quality software must address have been suggested for some time (e.g. Gilb, 1977). A curious aspect of these lists, explicit in the ISO/IEC 9126 standard (ISO/IEC, 1991), is that they typically only consider attributes of direct significance to the user. (This standard points out that there are several potential views of quality, including the user's view, the developer's view, and the manager's view, but only the user's view is in the current version, with others promised in later revisions.) See also (ISO, 1987e). Such a list, for instance, might be:

User Visible Aspects

- Appropriate functionality
- Coexistence and interoperability
- Ease of use
- Lack of surprises
- Adequate and usable documentation
- Ease of installation and update/cutover, including data.

A list of attributes of importance to a supplier would include these because keeping users satisfied is essential, but would go much further:

Supplier Visible Aspects

- Ease of learning for maintainers
- Ease of adaptability
- Structure for timeliness and cost-effective implementation
- Adequacy of exception handling
- Testability and measurability of product
- Analyzability and predictability of product
- Adequacy of scaffolding and support tools
- Professionalism of programming
- Efficiency and performance
- Ability to convince third parties of correctness, conformance, etc.?

3 MEASUREMENT OF PERCEPTIONS

The direct approach to measuring quality then is to study the perceptions that others have formed of the software, and extrapolate that to our situation. A central issue to address is whose opinions we want. In promoting subjective assessment of software products, we are in no sense suggesting that evaluation of software quality should be left to the intuition of the developers.

One obvious possibility is that of experts in the area. This is, effectively, what the refereeing process of journals provides, although the fact that referees reports are not made public diminishes the benefit third parties can gain from them. Most journals would welcome papers that are critiques and comparisons, but these are quite rare. Commercial publishers such as Ovum produce reports like this on popular topics, but they typically have limited accessibility due to price. An advantage of expert assessment is that experts are competent to know what are the potential strengths and weaknesses to look for, and how to study them. The disadvantage is that by knowing too much, the expert may not recognize obstacles that would impede the use of the software by novices, or by users coming from other disciplines.

Another possible source of opinions is journalistic reviewers, such as those published in the computer press. An advantage of this group is that they have professional incentive to do many reviews, and hence have a broad context against which to compare products. They also become adept at explaining their impressions of a product to a largely nontechnical audience. This however can also represent a disadvantage if a reader has a deeper understanding of the area than the reviewer addresses. Another disadvantage is that journalistic reviewers can be biased by their personal needs and experience, which are primarily journalistic not technical.

Yet another possibility, made practical largely by the newsgroups on the InterNet, is to assimilate the experiences of 'users like me.' Newsgroups such as comp.soft-sys.matlab, comp.soft-sys.math.mathematica, and sci.math.symbolic contain many items that provide insight into users' impressions of that particular software product: problems users have in appreciating how to use the software, creative ways the software can be used to perform complex tasks, desirable enhancements, etc. Items where a question posed draws responses from one or more other users, or even from supplier's representatives, are particularly interesting to third parties. Distilling quality evaluations from newsgroups is arduous, however, because of the flood of items, because the information is not presented in a form

suitable for automatic processing, and because the contributors do not represent a random sample (and indeed do not explicitly characterize themselves as to background, sophistication etc.). The situation could be improved if, rather than a newsgroup, a WWW website was used to collect such items in a database. Although HTML is too weak to facilitate mathematical notation directly, diagrams and even mathematical notation can be rendered by adroit use of GIF, which for mathematical software would be of real benefit over just using plain ASCII text. While a vendor's summary of such user feedback might be dismissed, the raw material being available means others, such as a user group or even an individual potential user, can provide their own analysis. Analysis of such data has much in common with retrospective statistical surveys, and while such studies do not enjoy the opportunities that prospective surveys have to use randomization to eliminate bias, there is a substantial literature on how to detect and ameliorate, if not correct for, its effects (Cochran, 1963; Stephan, 1958; Clark, 1991). Vendors might initially be sensitive to negative image possible from some of the postings, but the newsgroups contain those now, and on the whole are of net benefit to the products.

We are not suggesting that traditional 'objective' metrics be abandoned, but only that they should be appreciated in a different light. If the actual intent is to measure perception, measurement, however objective, of attributes of the software that we suspect might influence our perception, is an indirect approach. Relating these measured attributes to measured user satisfaction is surprisingly rare (Buckley, 1995). This indirect approach appears to have the attraction of being more quantitative and precise than the discursive presentation with checklists that typify the direct approach. It also appears to be less influenced by individual intuition and taste. There are, however, deeper considerations that need to be taken into account:

- The set of attributes to consider is problematical.
- The appearance of objectivity is somewhat misleading, in that many of the attributes are in fact qualitative, not quantitative.
- Even notionally quantitative attributes such as portability are not so in practice.
- For some attributes, the arbitrariness of any metric means numerical scores can distort the picture rather than refine it.
- Theorems proved about the software obviously increase our understanding of it, but may be of limited applicability unless the relevant conditions of the theorem can be readily established or the theorem can be shown to be robust.
- Ostensibly reproducible computational experiments can be carried out by the vendor, by an independent testing laboratory, or even by the potential user, to study well defined attributes such as accuracy, storage requirements or speed, yet batteries of tests have similar provisos.
- There is also the question as to whether problems studied by computational experiments should be representative of problems in the area, or should be illustrative of specific strengths (or weaknesses) of the software.
- Experimental assessment of representative tasks, such as extending the algorithm or integrating the software into a larger system, requires control on so many factors that the results are usually best understood as anecdotal.
- Some attributes are intrinsically difficult to observe, which often leads to instead studying surrogates that hopefully exhibit similar behaviour.

- A trap competitive developers often fall into is mistaking a score on the surrogate for the real objective.
- Interestingly, when our real purpose is to predict perception, it may not be necessary for the observed measurement to have a recognized causal relationship with quality at all, provided that there is an observed statistical correlation.

An even more indirect approach, currently in vogue, is to study the process by which the software was built. For custom software that has not yet been delivered, and for which there is hence no user community, it is not obvious what else can be done. The ISO 9000 standard family (ISO, 1987a; ISO, 1987b; ISO, 1987c; ISO, 1987d; ISO, 1987e; ISO, 1991)) basically requires that whatever process is chosen for development should be understood and documented and should be monitored to ensure it is actually used. The Software Engineering Institute's Capability Maturity Model, CMM, goes further in requiring what key areas process should address, and suggesting that the process should be measured so it could be optimized. Unfortunately, despite the considerable publicity over the past few years for process improvement, and despite the self evident nature of the assertion that process must make a difference, there is a dearth of quantified experimental scientific evidence that current methodologies have the claimed effects.

In summary, a quality assessment must be presented in such a way that in using the assessment your own judgment can be applied. An appreciation of how the software is intended to be used may be important in assessing its quality. An appreciation of who will be using it and how they will use it certainly is.

4 A SOUND PRECEDENT

Science is conservative and prefers to follow sound precedents. An interesting precedent to our point of view took place a decade ago in the field of acoustics, specifically, in the evaluation of loudspeaker quality. At the time, there was controversy about what were the appropriate metrics to use (Toole, 1986a). Over 50 years, technical measurement techniques had been developed for quality attributes such as amplitude response as a function of listening perspective, and such as phase response, transients, nonlinear distortions, and audibility of anomalies in amplitude-response and time-domain. The interpretation of these measured results is not obvious, as the results are typically not simple scalars but curves, and 'evidence that offends the eye may or may not indicate the presence of a problem that is offensive to the ear' (Toole, 1986a). Claims of relationship to better quality sound were mutually inconsistent: for instance a flat axial amplitude response across frequency mathematically cannot correspond to a flat frequency response with respect to total radiated acoustic power output and vice versa, yet each had proponents claiming superiority.

Of course listening tests and subjective preferences have always been used for evaluating audio equipment. However, there was a widespread belief that differences between listener tastes were so large, never mind the differences induced by different listening conditions, that there was real difficulty in defining the scope and validity of any result.

Toole developed a methodology for subjective experiments with thorough control of the acoustical, psychological, and experimental sources of variability (Toole, 1985). Using 42 listeners with a background of serious critical listening (from professional sound engineers to audiophiles, many of them musicians) he found that in comparative assessment of loudspeaker quality the variation in subjective preference ratings for those with normal hearing was less than 0.5 on a scale of 0 to 10. From these precise subjective measurements, he was able to go back to the technical measurements (Toole 1986b) and show which were consistent with the subjective measurements, which were inconsistent, where conventional data processing discarded significant information from the data, and where new types of technical measurements might be needed.

Recognition that perception was the fundamental criterion of quality thus not only led to better methodology for measuring it, but also improved the technology for technical measurements. Incidentally, this all led to significant improvements in commercial products. The long term objective of the project was achieved, i.e. to define a set of technical measurements and the form of their presentation such that interpretations by experienced unbiased observers corresponds to the results of controlled listening tests using unbiased listeners. Because designers find it easier to relate aspects of their designs to objective technical measurements than to subjective measurements, the improved technical measurements and the ability from them to predict subjective interpretation led to a new generation of loudspeaker designs perceived to have significantly higher quality audio.

5 PRODUCT REVIEWS

Although the idea of measuring quality by surveying users is novel in software engineering, it is not at all unusual in reviewing other products. Reviews are by experts, who rely for their credibility primarily on the reputation of the organization, not on their personal qualifications. Such reviews are published for many kinds of products: audio equipment, cameras, automobiles, and personal computer software, to name a few. We will consider briefly reviews of automobiles.

It is recognized that the perception of quality is so different for automobile enthusiasts and for those concerned only with safe, reliable, and economic transportation that not only are the reviews quite different, but they are published in different places: *Road and Track* and *Motor*, for instance, instead of *Consumers Reports* and *Which?* It is less well known that there is yet another set of reviews, published in the trade press of the service station business, considering the mechanics perceptions of difficulties and tricks for dealing with specific models.

Reviews for all these groups include specific lab measurements, such as turning radius, braking distance or fuel consumption, but the dominant parts of the reviews are the road tests and long term owner reports produced by expert drivers. A particularly valuable service provided by some reviews, such as Consumer Reports Annual Auto Issue (Consumers, 1996), is the compilation of owner surveys of reliability and other experience in use of alternative current and past products.

This kind of product review would be equally applicable, and equally valuable, for mathematical software.

6 CONCLUSIONS

We have argued that measuring quality is not just for quality assurance. We have suggested that it is wise to break free from narrow notions of what constitutes quality. From a user's perspective, we have indicated the importance of consideration of competition and of software lifetime over multiple release cycles. We have asserted that subjective assessment of quality can be useful, and that objective measures should be used to support subjective assessment.

REFERENCES

Buckley, M. and Chillarege,R. (1995) Discovering Relationships Between Coverage and Customer Satisfaction. *1995 Conference on Software Maintenance*, October 17–20, Opio, France, 192–201.

Carrol, J. M. and Rosson, M. B. (1984) Beyond MIPS*: Performance Is Not Quality. *BYTE*, 9(2), February, 168–72.

Clark, K. B., and Fujimoto, T. (1991) Product Development Performance. Harvard Business School Press, Boston

Cochran, W. G. (1953, 1963) *Sampling Techniques.* John Wiley and Sons, New York

Consumers Reports (1966) Annual Auto Issue, 61(4), April.

Galsworthy, John. (1912) *The Inn of Tranquillity and Other Essays and Studies*, Charles Scribner's Sons, New York

Gilb, T. (1977) *Software Metrics*, Winthrop, Cambridge, MA.

Gilbert, P. (1983) *Software Design and Development.* Science Research Associates, Inc., Chicago.

IEEE Std P1061–1988 Standard for a Software Quality Metrics Methodology.

IEEE Std 610.12–1990 Standard Glossary of Software Engineering Terminology (ANSI).

ISO/IEC 9126:1991(E) International Standard – Information Technology – Software product evaluation – Quality characteristics and guidelines for their use.

ISO/IEC 8402:1986 International Standard – Quality – vocabulary.

ISO 9000:1987 International Standard – Quality management and quality assurance standards – Guidelines for selection and use.

ISO 9000–3:1991 International Standard – Quality management and quality assurance standards – Part 3: Guidelines for the application of ISO 9001 to the development, supply and maintenance of software.

ISO 9001:1987 International Standard – Quality systems – Model for quality assurance in design/development, production, installation and servicing.

ISO 9003:1987 International Standard – Quality systems – Model for quality assurance in final inspection and test.

ISO 9004:1987 International Standard – Quality management and quality system elements – Guidelines.

Hailstone, R. (1991) Quality management and software engineering. in *Software Quality and Reliability, Tools and methods*, ed. Darrel Ince, Chapman & Hall, London.

Johnson, J. H. (1996) Private communication.

Kitchenham, B. A., and Walker, J. G. (1986) The meaning of quality. in *Proc. Conf. Software Engineering 86*, 393–406.

Oxford English Dictionary, Second Edition (1993), Oxford University Press, Oxford, UK.

Schach, S. R. (1990) *Software Engineering*. Aksen Associates Inc., Homewood, IL.

Stephan, F. and McCarthy, P. J. (1958), Sampling Opinions - An Analysis of Survey Procedures. John Wiley and Sons, New York

Tinnirello, P. C. (1995) *Handbook of Application Development, Second Edition*. Auerbach Publications, Boston.

Toole, F. E. (1985) Subjective measurements of loudspeaker sound quality and listener performance. *Journal of the Audio Engineering Society*, 33(1–2), Jan.–Feb., 2–32.

Toole, F. E. (1986a) Loudspeaker measurements and their relationship to listener preferences. I. *Journal of the Audio Engineering Society*, 34(4), April, 227–35.

Toole, F. E. (1986b) Loudspeaker measurements and their relationship to listener preferences. II *Journal of the Audio Engineering Society*, 34(5), May, 323–48.

NRC number: 40149

DISCUSSION

Speaker : W.M. Gentleman

M. Wright : In some recent discussions of "alternative medicine", there has been controversy because quality is measured by obviously subjective criteria, such as whether people believe the medicine makes them feel better. Do you believe that there are objective, measurable qualities in software that we as software builders must be concerned about? If so, what are they and how can we measure them?

W.M. Gentleman : The alternative medicine comparison illustrates some of the points I have been making. The patient, the epidemiologist, and the insurer all have very different perspectives, and it is hardly surprising if their subjective evaluations of a treatment might be different, even though for any one of them, the subjective assessment might be quite reproducible, and the subjective assessment might be very consistent across any such class of person. There is a role for an assessment of whether the medicine makes the patient feel better, it is just not the only valid assessment. An objective measure that could predict this would be useful. Similarly in assessing software quality, what we should ultimately be concerned with are subjective assessments. These determine whether the software will be accepted and used. Of the many possible objective measures, only those are really of interest that effectively predict subjective assessments. The relationship may not be simple: for instance, although we might objectively measure the tradeoff of time versus space for different versions of a given algorithm, the subjective assessment of satisfactory performance is likely to be influenced by the typical magnitudes of space and time requirements for the algorithm versus the available computing resources, and versus what is required for other activities. Appropriate objective measures are not always obvious. For example, often a problem can be formulated several ways, and although a mathematician will be attracted to the most general, most succinct, and most elegant formulation, this may not be the one users find easiest to work with. A simple subjective assessment of which formulation is preferred is not sufficiently informative to be helpful, but a more searching survey might reveal why some formulations are typically more effective than others. What objective measures might predict this subjective assessment? Perhaps some measure of the amount of work required to map the problem as it naturally arose into the formulation that the software addresses, or perhaps some measure of how much complication is introduced because the generality of the software ignores crucial aspects of the particular problem.

R. Boisvert : Serious product reviews for mathematical software are rarely done, but can be extremely useful. We have been formulating plans for incorporating informal reviews from users as an essential part of the refereeing process for the ACM Transactions on Mathematical Software. In particular, we are planning a network-based "community review" system for the Collected Algorithms. Under this system, algorithms would be posted in a public place upon receipt and announced in appropriate forums to solicit comment. A traditional track of expert refereeing would also be initiated at this time. Publication would occur only after both favorable public comment *and* favorable referee reports have been received. A system of quality ratings for published algorithms would

be established, and algorithms which continue to garner favorable public comment would be elevated to a higher rating based on the judgment of the editorial board.