Software quality: concepts and evidences

Luis Fernández Sanz
Departamento de Sistemas Informáticos
Universidad Europea de Madrid
luis.fernandez@uem.es

Introduction

The concept of software quality is more complex than what common people tend to believe. However, it is very popular both for common people and IT professionals. If we look at the definition of quality in a dictionary, it is usual to find something like the following: set of characteristics that allows us to rank things as better or worse than other similar ones. In many cases, dictionaries mention the idea of excellence together with this type of definitions.

Certainly, this idea of quality does not help engineers to improve results in the different fields of activity. In the world of industrial quality in general, a transition from a rigid concept to an adaptive one was performed many years ago. The concept view tend to be more close to the traditional idea of beauty: "it is in the eyes of the observer". So, we reject absolute concepts and tend to use customer satisfaction as main inspiration. For example, what characteristics are used by customers as indicators of "quality" (i.e. excellence):

- Product nature
- Reputation of raw materials: e.g. marble from Carrara.
- Manufacturing location: e.g. wine from La Rioja
- Manufacturing method: e.g. artisans
- Point-of-sale standing: why does the same beer cost much more at a sophisticated restaurant than at the usual pub.
- Price: "Quality is what expensive things have"
- Results: direct opinion transmission from one customer to another.

As Crossby told, quality is perhaps something different of what you think. He used the analogy between sex and quality (Crossby, 1980):

- Everybody agrees with it
- Everybody thinks he/she knows everything about it
- Everybody thinks it is enough to follow his/her intuition
- Everybody thinks problems are others' fault

Software quality

Different attempts to define software quality as a complex concept that can be decomposed in more detailed characteristics have been presented since 1970s, e.g. (McCall et al, 1977) (see Figure 1). The idea was to enable evaluation of quality through the evaluation of more detailed characteristics that are supposed to be easy to measure or assess. The problem with this work line is that such features are not easy to evaluate in a subjective manner or, even, they are not clearly defined. Standardized quality models based on this idea, e.g. ISO 9126 (ISO,), are only useful as source of ideas to establish an agreement for better understanding between customer and developer because it is not clear what kind of metrics are really validated and feasible for a correct measurement of each characteristic. For example, metrics proposed by Mc call et al. were clearly obsolete, not validated or simply subjective.

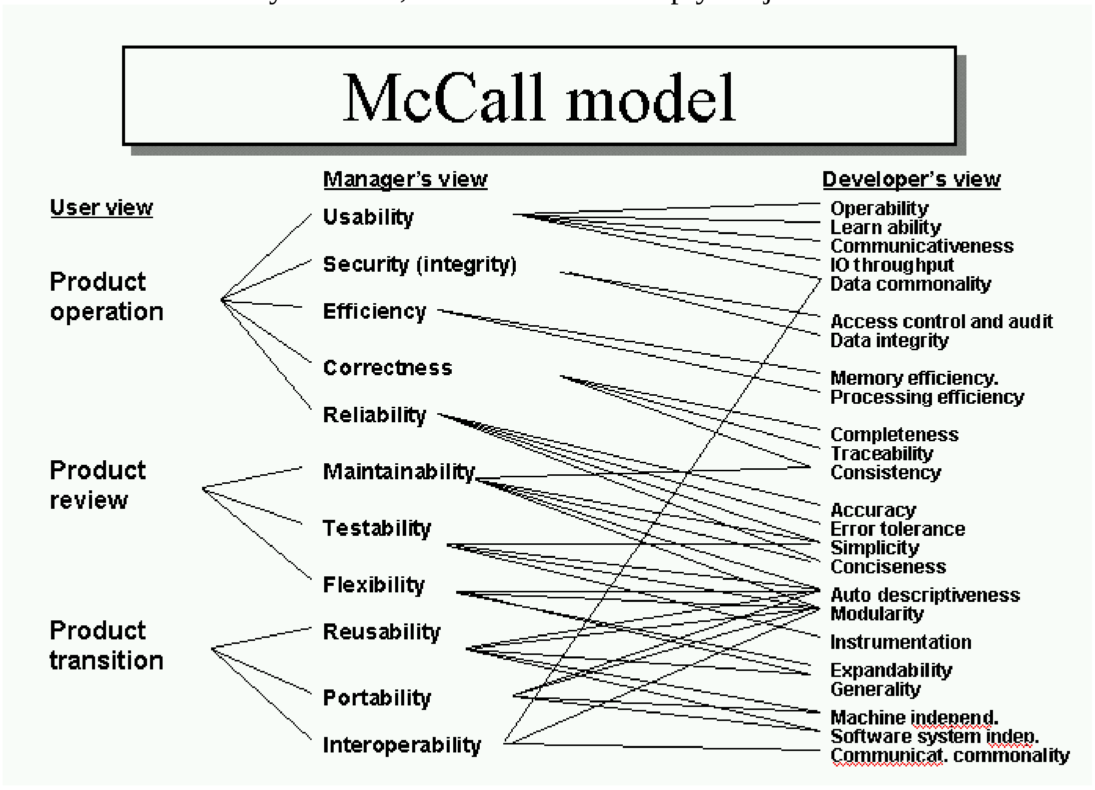

Figure 1. Decomposition tree of McCall et al. software quality model

Moreover, the different factors or attributes are not independent. For example, high values in portability tend to cause low levels of efficiency. There are different positive (reinforcement) and negative (counteracting) relationships clearly identified among the different attributes in this type of models. The actual trend is the definition of adaptive models for measuring software quality like GQM (Basili, 1988) or COQUAMO (Kitchenham & Walker, 1989).

Nonetheless, it is usual that real software projects have not enough resources and/or efficient planning processes to afford such a big number of different values to be monitored. In these cases, organizations prefer to use a more limited concept of software quality based on detecting its absence in a system: metrics are based on different variations of the basic number of detected defects in the software application. This concept of software quality is well accepted by practitioners and organizations because data to be measured is usually collected during project and maintenance activity (e.g., defects reported by customers, defects detected during testing, changes controlled by configuration management system, etc.). Defect rates ranking is very popular and organizations tend to consider what are the standards in industry and which is its position (e.g. higher or lower than 1 defect/KLOC). As (Adams, 1984) found out, one of the problems of this approach is the fact that not all the defects lead to quality problems (i.e. failures): up to 1/3 of the defects after 5000 years of system functioning.

Proposed lines of action

When quality has to be measured, it is important to follow the foundations of every measurement activity to avoid mistakes. Measurement can be defined as “the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules” (Fenton & Pfleeger, 1997). Attributes can be internal (measurable in terms of the entity itself) or external (measurable only with respect to how the entity relates to its environment). Software entities may refer to processes, products or resources. Allocation of numbers and symbols can be done in a subjective (mainly based on opinions) or objective way; it is also possible to distinguish between direct (involves no other attribute or entity) and indirect measurement (e.g. defect rate = num.defects/size).

What it is extremely important for assuring appropriate measurement activity is to have a clear idea of what are the expected objectives. So, do we think that IQ is really a real measure of intelligence or just an indicator of ability to answer specific tests? In the case of software, is found defects/time a metric of software quality or is it an indicator of quality of testing? Is LOC/time a measure of productivity or of speed of coding?

As someone said “evaluate is to measure; but although a cm is an objective unit it is not the same to measure your body to order a dress to a tailor compared with measuring it to order your coffin”. It is also important to be aware of the Hawthorne effect, i.e., the possibility that expectations tend to alter the collected values due to a change in the normal behavior of people who know that is observed.

Having clear objectives helps us to define valid metrics. Validity has not always been preserved even in the case of very popular metrics. Basically a valid metric is the one for which it is shown that really measures what it claims. One of the basic elements for that is the existence of well defined rules of calculating it. For example, depending on what definition we adopt for line of code, the same piece of code can experiment differences up to 5:1 in the number of LOC.

One promising work line for software measurement is the conceptualization of the measurement validity criteria as the preservation of the empirical relationships between entities in the real world when values are computed (Fenton & Pfleeger, 1997). Using this approach, contributions like the one of (Briand et al., 1996) where basic properties of software (like complexity, size, etc.) are characterized to enable the control of validity of the measures that claim to measure such properties of a software product.

Conclusions

Different ideas about the concept of software quality and how to evaluate it has been presented in this paper. A brief review of different proposals and contributions in this area has been also reviewed to offer an overall image of the main challenges to be addressed. As a final point of interest, I think that valid and solid measurement is essential for software engineering advance because there is a short tradition on justifying by empirical means the benefits of proposals. For example, process models and standards like CMMi or ISO 9001 have not always demonstrated their value for reaching benefits like productivity, quality, etc. Everybody assume that better process leads to better product results (by the way, I am sure this is true) but are there enough and solid set of real data to support it? Of course, CMMi, at least, has tried to measure benefits and quantify the relationship between efforts in process side and benefits. Moreover there are other (¿psychological?) benefits: stabilization of processes, etc. The same reflection applies to the assumption that better structure leads to better external behavior. It is not always a clear relationship.

In summary, I strongly believe that rather than looking for or “buying” silver bullets, we should try to explore concepts behind commonly adopted practices in industry to be sure that those practices are the best ones (or, at least, beneficial).

References

Adams, E. (1984). Optimizing preventive service of software products. IBM Journal of Reserch and Development. 28 (1). 2-14.

Basili, V.R. and Rombach, H.D. (1988). The TAME Project: towards improvement-oriented software environments. IEEE Transactions on Software Engineering. 14 (6), 758-773.

Briand, L.C., Morasca, S. y Basili, V.R. (1996) "Property-based software engineering measurement". IEEE Transactions on software engineering. 22 (1), 68-85.

Crossby, Philip B. (1980). Quality is Free: The Art of Marketing Quality Certain. New York, Mentor Books.

Fenton, N.E. and Pfleeger, S.L. (1997). Sofwtare metrics. A rigorous and practical approach. London, PWS.

ISO (1991), Software product evaluation. Quality characteristics and guidelines for their use, Geneve, author.

Kitchenham, B. A. and Walker, J. G. (1989). A quantitative approach to monitoring software development. Software Engineering Journal. 4 (1), 2-13.

McCall, J. A., Richards, P. K. and Walters, G. F. (1977). Factors in Software Quality, Volumes I, II, andIII, US. Rome Air Development Center ReportsNTIS AD/A-049 014, NTIS AD/A-049 015 and NTIS AD/A-049 016, U. S. Department of Commerce.