Software quality measurement

Magne Jørgensen
University of Oslo, Department of Informatics, Norway

Abstract: This paper analyses our ability to measure software quality. The analysis is based on the representational theory of measurement. We analyse three assumptions on software quality measurement and recommend an increased measurement focus on establishing empirical relational systems (models of what we measure) in software quality related work. Applying measurement theory on the measurement of software quality related properties means that we have a powerful tool to understand what we measure, what the meaningful operations on the measured values are, and how to interpret the results.

1 Introduction

Most of the software quality standards and frameworks, such as ISO 9001/9000-3, the Capability Maturity Model[11], ANSI/IEEE Std. 730-1989 and ESA PSS-05-0 1991, require or recommend measurement of software quality. Unfortunately, there is a large gap between the requirement **that** quality measurement should be carried out and the guidelines on **how** to carry out the measurements. For example, the software quality standard ISO 9000-3 (which is the guideline for the use of ISO 9001 on software production) states in Section 6.4.1 that: *"There are currently no universally accepted measures of software quality. ... The supplier of software products should collect and act on quantitative measures of the quality of these software products."*

Here, ISO 9000-3 requires quality measurement and at the same time admits that there are no (universally) accepted quality measures. In order not to have contradicting requirements ISO 9000-3 (and other similar frameworks) may have made one or more of the following assumptions:

A1: Although no universal software quality measures exist, there are meaningful quality measures for particular environments.

A2: Widely accepted quality measures will occur, when the software quality measurement research becomes more mature.

A3: Measurement of software quality indicators (so-called quality factors) can be measured and used to predict or indirectly measure the software quality.

This paper analyses the validity of these assumptions from the perspective of the "representational measurement theory".

We give a brief introduction to measurement theory in Section 2. In Section 3 we apply the theory to formulate the necessary preconditions for meaningful measurement of software quality. Then, in Section 4, we discuss whether common quality definitions enable meaningful measurement of software quality. Finally, in Section 5, we draw conclusions about the validity of the assumptions A1-A3 and recommend actions based on the analysis results.

1.1 Previous work on measurement theory and software quality

Possibly, the first paper on measurement theory was written by Stevens[15] in 1946. In 1981, the first paper connecting software measurement and measurement theory was published, see DeMillo and Liption[6]. The first real application of measurement theory on software measurement was, according to Melton[10], not carried out before the late 1980s. In the 1990s, there have been several software measurement theory papers, mainly on the evaluation of software complexity measures, see for example Zuse[16] and Fenton[7]. As far as we know, no paper on measurement theory in depth analysing software quality measurement has been published.

2 A brief introduction to measurement theory

When we measure length in meters and weight in kilograms we do not reflect much on what we do when we measure. Measuring an attribute where we have a common understanding of what we measure and accepted measurement methods, this lack of reflection is not harmful. Measuring software quality, however, we do not have an accepted understanding of what quality is, nor do we have commonly accepted software quality measures. For this reason, we need a framework for quality measurement reflections, such as reflections on:

- the preconditions for quality measurement
- how to analyse the meaningfulness of quality measures, and
- how to interpret and use the measured values.

We argue that measurement theory establishes such a framework.

2.1 Measurement theory definitions

The following definitions were adapted from Melton[10]:

Def. Empirical Relational system: $<E,\{R_1..R_n\}>$, where E is a set of entities and $R_1..R_n$ the set of empirical relations defined on E with respect to a given attribute (for example, the attribute quality).

The empirical relational system is a model of the part of the "real world" we are interested in, i.e. a perspective on our knowledge about the phenomenon to be measured. The model should ensure agreement about the empirical relations and enable measurement. For example, to state that

program A has more modules than program B we need a model of programs that enable us to agree upon what a module is and how to identify it.

Def. Formal (numerical) Relational system: $<N,\{S_1..S_n\}>$, where N is a set of numerals or symbols, and $S_1..S_n$ the set of numerical relations defined on N.

Def. Measure: M is a measure for $<E,\{R_1..R_n\}>$ with respect to a given attribute iff:

1. $M: E \rightarrow N$
2. $R_i(e_1, e_2, ... e_k) \Leftrightarrow S_i(M(e_1), M(e_2), ... M(e_k))$, for all i.

Condition 1 says that a measure is a mapping from entities to numbers or symbols. Condition 2, the representation condition, requires equivalence between the empirical and the formal relations.

Def. Admissible transformation: Let M be a measure, E a set of entities, N a set of numerals and F a transformation (mapping) from M(E) to N, i.e. $F: M(E) \rightarrow N$. F is an admissible transformation iff F(M(E)) is a measure.

In other words, an admissible transformation preserves the equivalence between the empirical relations and the formal relations.

The definition of admissible transformations enables a **classification of scales**. A common classification of scales is the following:

Nominal scale: Admissible transformations are one-to-one mappings. The only empirical relation possible is related to "equality". *Separating programs into "Fortran-programs" and "Cobol-programs" leads to a nominal scale.*

Ordinal scale: Admissible transformations are strictly increasing functions. The empirical relations possible are related to "equality" and "order". *Assigning the values "high quality", "medium quality" and "low quality" to software leads to an ordinal scale.*

Interval scale: Admissible transformations are of the type $F(x) = ax + b$, $a > 0$. The empirical relations possible are related to "equality", "order" and "difference". *The temperature scale (in degrees Celsius) is an interval scale.*

Ratio scale: Admissible transformations are of type $F(x) = ax$, $a > 0$. The empirical relations possible are "equality", "order", "difference" and "relative difference". *The length of programs measured in lines of code forms a ratio scale.*

We believe that software quality should at least be measurable on an **ordinal scale**. Otherwise, we would not be able to state that software A is "better" than software B with respect to quality, i.e. our intuition of what software quality is would be strongly violated. In measurement theory terminology this means that we should require that the empirical relational system includes an accepted understanding of the relation "higher quality than".

The appropriateness of statistical and mathematical methods to the measured values are determined by the type of scale, i.e. by the admissible

transformations on the scale values. For example, addition of values are meaningful for values on an interval or ratio scale, not on an ordinal or nominal scale. This means, for example, that “units of software quality” must be meaningful to calculate the mean software quality. There are diverging opinions on how rigidly the scale prescriptions should be interpreted, see Briand et al.[2] for an overview.

3 Preconditions for the measurement of software quality

Dependent on how we define software quality, software quality may be **directly** measured, **indirectly** measured or **predicted**. The preconditions for the different measurement types differ. Below we have applied the measurement preconditions on the measurement of software quality.

The preconditions for **direct measurement** of software quality are that:

1. The empirical relational system of software quality is established. In our opinion, this means that, at least, we should have a common understanding of the relations “same quality as” and “better quality than”, i.e. we must be able to decide whether software A has "better", "same" or "worse" quality than software B.
2. A numerical or symbolic system with equivalent formal relations to the empirical quality relations is established, for example the formal relations “=“ and “>“ representing the relations "same" and "better" quality in the empirical relational system.
3. A measure (mapping) from the attribute software quality to numbers or symbols is defined.
4. A measurement tool or method implementing the measure is implemented.

The preconditions for **indirect measurement** of software quality are that:

1. The preconditions for measurement of the directly measured software attributes are met.
2. A complete empirical connection between the directly measured attributes and the indirectly measured software quality is established. The connection is in our case complete when all aspects of the agreed understanding of software quality can be determined from the directly measured attributes.
3. The connection is accurately translated into the formal relational system (through a formula).

Predictions of software quality are similar to indirect measurement. The difference is that predictions do not require a complete empirical connection or an accurate translation into the formal relational system, i.e. all the empirical relations are not necessarily preserved into the formal relational system.

4 Common quality definitions and measurement theory

The most common types of software quality definitions are, probably, the following three:

1. Software quality is determined by a set of quality factors (see for example ISO 8402-1986 and IEEE 610.12-1990).
2. Software quality is determined by the user satisfaction, see for example Deephouse[5].
3. Software quality is determined by the errors or unexpected behaviour of the software, see for example Carey[3] and Lanubile[9].

These definitions seem to be based on a similar "intuition" of what software quality is. For example, they seem to share the opinion that software quality is the degree of meeting the user needs. The difference may lay in whether they consider user needs in the form of (1) requirements and implied needs, (2) user satisfaction or (3) the level of incorrect behaviour of the software. From the viewpoint of measurement theory these differences are significant, and lead to different empirical relational systems.

Notice that there are numerous other definitions and perspectives on (software) quality, see for example Garvin[8] and Dahlbom & Mathiassen[4] for classifications and overviews.

4.1. Quality as a set of quality factors

An example of "quality as a set of quality factors" is the definition in ISO 8402-1986 which defines quality as: *The totality of features and characteristics of a product or service that bear on its ability to meet stated or implied needs.*

Examples of such "features and characteristics" (quality factors) are efficiency, flexibility, integrity, interoperability, maintainability and portability.

The above type of definition has, among others, the following implications for the empirical relational system of software quality:

- The relation "better quality than" should be interpreted as a better ability to meet stated and implied needs.
- In order to measure and compare software quality we need to formulate an empirical connection between the quality factors and the software quality itself, i.e. software quality is **indirectly** measured.

We will argue that the preconditions for indirect measurement are very hard to meet for this type of empirical relational system for the following reasons:

- It is far from obvious that all the quality factors, for example maintainability, are measurable. In our opinion, some of the quality factors are just as difficult to measure as the software quality itself.
- As long as our understanding of software quality is at an "intuitive" level and not explicitly formulated we will not be able to establish a complete empirical connection between the quality factors and the software quality. For example: A*ssume that we believe (or agree on) that the only relevant quality factors are reliability and usability, and that these quality factors are measurable. We will probably not always be able to agree on the total impact on the quality of a decreased usability together with (or perhaps*

caused by) an increased reliability. In other words, we have no complete connection between quality and the quality factors reliability and usability.

From a measurement theory viewpoint, this means that the quality factor based definitions do not enable measurement of software quality. One might argue that the quality factor definitions suggest a measurement of quality factors in order to **predict** the software quality. In this situation an accurate connection between the quality factors and the software quality is not needed. On the other hand, in order to have a meaningful prediction system we still need an empirical relational system including an accepted understanding of the relation "higher quality than", i.e. an accepted understanding of "a higher ability to meet user needs". Currently, such an empirical relational system does not exist.

4.2 Quality as user satisfaction

A common approach, for example frequently applied in the quality framework Total Quality Management (TQM), is to define or understand software quality as the level of user satisfaction. This understanding of software quality has, among others, the following consequences for the empirical relational system of software quality:

- there must be a commonly accepted method of identifying the user satisfaction
- there must be a commonly accepted meaning of the relations "same quality as" and "better quality than" based on user satisfaction

We may argue that this is possible because:

- a method of identifying the user satisfaction may be to ask them
- "same/higher quality than" may be understood as the same/higher proportion of satisfied users.

In addition, a measure (mapping) from software quality to numbers or symbols can be the mapping from "empirical proportions" to "numerical proportions" of satisfied users. This measure preserves the relation "higher quality than".

The main problems with this type of software quality measurement are, in our opinion, that:

- Why call it software quality, when we measure user satisfaction?
- If all types of "users" (such as the maintainers and the "end users") should be considered, it will be difficult to establish a complete connection between the different types of user satisfaction values and the (total) software quality.
- How do we know that user A's "very satisfied" means the same as user B's "very satisfied"?

4.3 Software quality related to errors

In spite of the high number of sophisticated definitions of software quality, the most common software industry interpretation of software quality may be "the degree of errors in the software". In order to meaningfully compare this type of software quality among software products with different sizes, the number of errors gets divided by the software size. Software size may, for example, be measured in "lines of code".

IEEE 1044-1993 defines an error (anomaly) as *any condition that departs from the expected*. The expectations may be described in a document or be based on other sources. To count (and categorise) errors we need to agree on how to decide whether a condition is expected or not, and whether a condition should be counted as one or more errors. To some extent, this seems to be possible. For example, we could agree that only the expectations specified in the requirement specification and the obvious implications of the requirement specifications should be counted as expectations.

Surprisingly, there is no commonly accepted definition of lines of code. There are, however, many company specific definitions which enable measurement of lines of code. Lines of code is a frequently criticised measure. The criticism, however, has mainly been on the use of lines of code in measurement of productivity. From a measurement theoretical viewpoint lines of code can be a valid measure of the “physical” size of the source code of software.

Assuming that we accept that software quality can be measured as number of errors divided by software size, the relation “higher quality than” may be interpreted as a lower value of the ratio “errors/lines of code”.

A naive use of these measure, i.e. not distinguishing between different types of errors, may give counter intuitive results. For example, two cosmetic errors have a stronger impact on the quality than one serious error.

To avoid this counter intuitive result, a classification of the errors in, for example, severity of failure (see IEEE 1044-1993 for an overview of classifications) is needed. This means, however, that we need a complete empirical connection between the error categories and the software quality in order to measure quality, i.e. indirect measurement. In practice, this type of indirect measurement seems to be just as difficult as the indirect measurement through the quality factors.

5 Conclusions and recommendations

Based on the previous analyses, we now examine the assumptions from Section 1:

A1: *Although no universal software quality measures exists, there are meaningful quality measures for particular environments.*

In order to establish meaningful software quality measures for particular environments and situations the involved persons should have approximately the same empirical relational system of software quality. For example, the users may decide that the only software characteristic important for the quality is how much the software decreases the error density in registering customer orders. Assuming a baseline error density the software quality may be measured as the percentage decrease in error density. From a measurement theory viewpoint this is an example of valid measurement. It is, on the other hand, an example of a label on a measure that hides what is actually being measured.

A lot of empirical studies on software development and maintenance define quality measures for particular environments (or applications), see for example the quality measures of the Software Engineering Laboratory for the NASA Goddard environment described in Basili & Selby[1]. The variation of the empirical relational systems of software quality in these studies is very high. An example is Schneiderman[13] who uses software quality as a label on a measure of the ease of memorisation/recalling program information.

For this reason, the reader of software quality studies should always try to get an understanding of the empirical relational system of a software quality measure before interpreting the results. When there is no agreed understanding of software quality, or only intuitive and non-articulated connections between what is measured and software quality, the measurement is not a meaningful quality measurement.

Conclusion: Assumption A1 cannot be rejected. It is, however, not a good idea to label a measure "software quality" if the measure represent an empirical relational system that is far from our intuition of what software quality is. Environment specific software quality measures seem, frequently, to be of this type and a lot of misunderstanding can be avoided using other labels on the measures.

A2: *Widely accepted quality measures will occur, when the software quality measurement research becomes more mature.*

If we require that the empirical relational system shall completely correspond with our intuitive understanding of software quality, it is not likely that we will see widely accepted measures of software quality. This lack of ability to agree on the understanding of software quality has, in our opinion, not much to do with the maturity of the software measurement discipline. Instead it has to do with the intrinsic difficulties of explicitly formulating what we understand by the very complex phenomenon “software quality” and the many different context and culture dependent viewpoints on software quality.

It is, of course, possible to "standardise" on one or more viewpoints on software quality - which has been done by many international standards and frameworks for software quality (see previous sections). Unfortunately, we may have to choose between a vague quality definition, which does not describe a complete empirical relational system enabling software quality measurement, and a well-described empirical relational system, which does not correspond to the intuitive understanding of software quality.

The wish expressed in Price Waterhouse[12] *".. research should be undertaken with a view to developing a workable definition of software quality, and measures of quality... that can be easily implemented by software developers."* may therefore be hard to meet if aiming at widely accepted quality measures.

Conclusion: If we want widely accepted software quality measures, we need to standardise on how to interpret software quality. This may be possible to achieve within the software quality measurement research community. It seems, however, unrealistic to expect that the software industry will adopt such a standardisation of software quality.

A3: *Measurement of software quality indicators (so-called quality factors) can be measured and used to predict or indirectly measure the software quality.*

The method of dividing a complex characteristic into less complex characteristics in order to indirectly measure the complex characteristic is typical for scientific work, and may have inspired for example IEEE in their software quality factor work, see Schneidewind[14]. This type of indirect measurement is only meaningful if an empirical connection between the directly and the indirectly measured characteristics is established. This type of connection is, for example, established for the indirect measurement of "speed" through the direct measurement of "length" per "time unit". However, in spite of a lot of effort we have not been able to agree on similar connections for software quality. Even worse, it is not obvious that the most common quality factors, such as maintainability and user friendliness, are less complex to measure than quality itself.

Conclusion: As long as we are not able to establish empirical connections between the quality factors and the quality itself, the quality factor work may not be able to contribute much to enable measurement of software quality.

On the basis of the previous analysis, we **recommend** that:

- Instead of trying to develop a widely accepted software quality measure (or set/vectors of quality factors) we should focus on establishing empirical relational systems and better measures for the more measurable quality related properties of software, such as user satisfaction or error density.

Measurement theory should be applied in order to know what we measure and how to interpret the results.

- The measures of software quality related attributes should not be called software quality measures. For example, to call a measure of error density a quality measure is unnecessary and misleading.
- There should be an increased empirical research and industry focus on the connections **between** the different quality related attributes. Measurement theory should be consulted when describing these connections. This way we may be able to build better prediction systems for quality related attributes, such as user satisfaction or reliability.

Index: measurement theory, software quality, quality measures.

References:

1. Basili, V.R., & Selby, R.W. Calculation and use of an environment's characteristic software metric set, pp. 386-393, *Proceedings of the 8th Int. Conf. on Software Engineering*, London, 1985.
2. Briand, L. et al. On the application of measurement theory in software engineering, *Empirical Software Engineering*, 1996, vol 1, no 1.
3. Carey, D. Is "Software quality" intrinsic, subjective or relational, *Software Engineering Notes*, 1996, vol 21, no 1, pp. 74-75.
4. Dahlbom, B. and Mathiassen, L. *Computers in context*, NCC Blackwells, Oxford, 1995.
5. Deephouse, C. et al. The effects of software processes on meeting targets and quality, pp 710-719, vol. 4, *Proceedings of the Twenty-Eighth Hawaii International Conference on System Sciences*, Hawaii, 1995.
6. DeMillo, R.A. & Liption, R.J. Software project forecasting, pp. 77-89, *Software Metrics*, Perlis, A.J. et al. (eds), Cambridge, MA: MIT Press, 1981.
7. Fenton, N. Software measurement: a necessary scientific basis, *IEEE Transactions on Software Engineering*, vol 20, no 3, 1994, pp 199-206.
8. Garvin, A. Competing on the eight dimensions of quality, *Harvard Business Review*, Nov-Dec 1987.
9. Lanubile F. & Visaggio, G. *Evaluating predicting quality models derived from software measures: lessons learned*, Technical report CS-TR-3606, Dept. of Computer Science, University of Maryland, 1996.
10. Melton, A. (ed) *Software Measurement*, International Thomson Computer Press, 1995.
11. Paulk, C. et al., *The capability maturity model: guidelines for improving the software process*, Addison-Wesley, 1995.
12. Price Waterhouse, *Software quality standards: The costs and benefits*, Price Waterhouse, London, 1988.
13. Schneiderman, B. Measuring computer program quality and comprehension. *Int. J. Man-Machine Studies*, vol 9, 1977, pp. 465-478.

14. Schneidewind, N. Methodology for validating software metrics, *IEEE Transactions on software engineering*, vol 18, no 5, 1992.
15. Stevens, S. On the theory of scales of measurement, *Science*, vol 103, pp. 677-680, 1946.
16. Zuse, H. *Software Complexity: Measures and Methods*. Amsterdam: de Gruiter, 1990.