An Investigation of Desired Quality Attributes for Different Types of Software

Tim Musson[a] and Ed Dodman[b]

[a]Department of Computer Studies, Napier University, 219 Colinton Road, Edinburgh, EH14 1DJ, UK
[b]Faculty of Inf & Eng Systems, Leeds Metropolitan University, The Grange, Beckett Park, Leeds, UK

Abstract

The first part of this paper gives an overview of some theoretical issues in software quality: quality models and different approaches to evaluating quality. The second part presents a questionnaire based investigation, concerning both the hypothesis that different software types will have specific profiles of quality attributes and current practice relating to quality attributes, their assurance and their measurement. While failing to confirm the hypothesis the questionnaire allows some interesting observations to be made in relation to the theoretical issues discussed earlier.

Keyword Codes: D.2.1; D.2.8
Keywords: Software Engineering, Requirements/Specifications; Software Engineering, Metrics

1. INTRODUCTION: THEORETICAL ISSUES IN SOFTWARE QUALITY

Current approaches to assuring and measuring software quality are predominantly process based rather than product based. There are two major assumptions here:

1) if we look after the process the product will look after itself

2) quality must be built into the product (can only be assessed when complete and difficult to alter).

These approaches allow the delivery of poor quality software in spite of a seemingly wonderful quality management system: the process functions smoothly, delivering a product on time, within budget and with all appropriate documents approved and signed. However the product may function far from smoothly, rating badly on a variety of quality attributes. Probably the main reason for this emphasis on procedure is that the software product is notoriously difficult to assess for quality whereas the process appears much more amenable. In spite of this, official standards for quality management systems, such as BS 5750, require procedures aimed at continually improving both process and product quality. In order to do this in any meaningful way it is necessary to be able to assess product quality.

1.1 Quality Models

Software quality models are principally of two types: general and specific. Two influential general models are those of McCall et al [1] and Boehm et al [2]. These treat quality as an attribute made up of high level "quality factors" (mainly external attributes such as maintainability, efficiency, reliability, usability) which are decomposed to give "quality criteria". These are associated with low level directly measurable "quality metrics". Factors and decompositions are assumed fixed for all software products.

The problems associated with these fixed general quality models include:

1) a standard set of quality factors and their decomposition is not suitable for all types of software

2) factors, criteria and metrics are of very diverse types (e.g. quality metrics are a mixture of genuine metrics, checklists of design procedures and production standards) [3]

3) managers in different environments may wish to interpret a particular factor in different ways

1.1.1 Gilb's Evolutionary Development and Kitchenham's COQUAMO

More recently general software quality models have been extended by Gilb [4] and Kitchenham [3]. Gilb's model is based on the notion of evolutionary development and delivery together with "measurable objectives": the system is delivered incrementally, always adding the functionality which is likely to give the best value for effort expended and ensuring that the measurable objectives are met. The importance of different aspects of functionality, together with the specification of non-functional attributes (the measurable objectives) are decided by the software engineer and the user in consultation. These non-functional attributes are quality attributes. Gilb [4] argues that they should be decomposed into low-level attributes, with fairly simple measurement procedures. For each of these low-level attributes it is necessary to define: (1) the unit of measurement; (2) the desired value of the measure; (3) the method of objectively determining a value on the scale; (4) the worst acceptable value of the measure; (5) the best known value of the measure anywhere; (6) the current level of the measure, for comparison purposes. This clearly assumes that any desirable attribute must be objectively measurable.

Kitchenham's [3] work attempts to combine Gilb's approach with the earlier work of McCall and Boehm. Her work concentrates on the development of a constructive quality model, "COQUAMO", and also attempts to relate process attributes to objective software measures. This work extracted the following quality factors from the above models:

efficiency, reliability, usability, extendibility, generality, testability, understandability, re-usability, maintainability, correctness, integrity, survivability.

Each of these factors is then classified as one of:

application-specific,	requiring application-specific definition,
general with application-independent definition,	software production related.

Thus some factors are considered to be totally general and form part of every model. The rest of the model is constructed from factors in other categories: the selection and, possibly, definition of these is carried out in consultation between the software engineer and the user.

1.1.2 Other General Software Quality Models

Delen and Rijsenbrij [5] have proposed a general model specifically for information systems: this restriction made quality assessment more tractable. Their model is based on decomposition, with the top-level product-oriented factors categorised as either static or dynamic. The bottom-level "quality attributes" (quality metrics) are characterised by:

definition	means of specifying requirements in verifiable fashion
relevant engineering actions	units (e.g. person hours)
measurability (explicit direct, implicit direct, indirect)	

The quality attributes are specified by the model, but the requirements for each are specified by the user. While Delen and Rijsenbrij make no claims concerning the success of their approach it seems very reasonable for a restricted domain such as information systems.

A recent development in this area, while not in itself a quality model, is the use of quality function deployment (QFD) [6]. This tool has been well-described in the general quality literature (e.g. [7]) but very little in the software quality literature. In the context of software development QFD facilitates the mapping of customer requirements via software characteristics onto production features. Erikkson and McFadden claim that QFD gives a good structure within which the users' needs are taken into account and progressively refined to give appropriate product metrics. One interesting aspect of this approach is that it implies that development methods should be adapted to the quality attribute requirements.

1.2 Specific Quality Models

A variety of specific quality models have also been proposed. These define quality in terms of one factor, which is decomposed into one or more measurable attributes and usually based on the number of defects found in a software product. The main aim here seems to be to ensure that contractual obligations are met: this coincides with one of Garvin's [8] major approaches to quality - manufacturing-based ("conformance to specification"), whereas the models discussed above relate more to his other major approach - user-based ("fitness for purpose"). This type of approach can assess one aspect, possibly very important for some systems, of software quality. However, it is unlikely that for a complex and multi-dimensional software system, it is best to base quality evaluation on one factor.

AT&T Bell Labs have used such an approach [9]. A variety of measures were used, but all were based on the number of faults found. Such a set of metrics is more a process than a product measure. If many faults are found and fixed this may mean that we now have a very reliable product, or a badly designed, badly constructed product, or both (and fixing the faults may have caused the design to deteriorate further). Even with respect to the process it is not totally clear what these metrics tell us.

Another example of this type of approach is given by Daskalantonakis [10]. He describes the set of metrics used for software evaluation in Motorola. Again, most of these measure defects and failures, with a few specifically relating to the process (e.g. productivity, project duration). The metrics were developed using the Goal/Question/Metric (GQM) approach. A set of goals were established and, for each of these, a set of questions determined, which would allow achievement of the goal to be assessed. Each question was answered using one or more metrics. For example:

Goal 3: Increase Software Reliability

Question 3.1: What is the rate of software failures, and how does it change over time?

Metric 3.1: Failure rate (FR) = number of failures / execution time

Even though this approach seems very restrictive Daskalantonakis claims very substantial benefits from the implementation of the metrics: "There are also many indirect benefits ... improve ship-acceptance criteria, and schedule estimation accuracy... Another long range benefit (which has been actually achieved so far within Motorola) is significant cost reduction due to improved quality". The benefits may well have been even greater if a wider-ranging set of metrics had been used.

An often ignored aspect with this type of approach is the user. For interactive systems it is essential this does not happen: the quality of interaction between the system and the user all important. No amount of internal 'structural' quality or process quality will make up for a lack of external 'functional' quality. The questionnaire results, presented below, give some interesting conclusions with respect to this.

2. THE QUESTIONNAIRE

The above discussion shows that the assessment of product quality is usually based on a generic set of factors applicable to all software. Musson [11] proposed that this could be inappropriate for software and that it might be possible to factor out differing quality requirements for differing software types.

A questionnaire (see Appendix A) was developed to test this hypothesis and to collect information on the actual development processes currently used inside the software industry, the quality attributes considered important and the means used to assess and achieve desired levels of these.

The first group of four questions was designed to locate the responding companies within the software production sector and to define the type of software they produce. The next three questions, relating to the design methods and techniques, computer languages and software tools used, represent an indirect approach to finding out what design paradigms are being utilised.

The next three questions related directly to the quality attributes important to the software producers, how they are assessed and how the achievement of these attributes is measured. Finally respondents were invited to introduce other relevant quality matters ; in general there was very little response to this.

One hundred and forty six questionnaires were sent to firms selected from three sources which were considered to include appropriate companies.

The questionnaire was, wherever possible, mailed to named software managers. Where names were unknown the questionnaires were mailed to the Software Manager or the nearest equivalent title that could be found for the company.

A covering letter explained that the purpose of the questionnaire was to gather information about current industrial thought and practice relating to quality and that all companies responding would receive a full report of the findings of the survey. To ensure anonymity it was made clear that the return slips with names and addresses would be immediately separated from the returned questionnaires.

The questionnaire was designed so that it could be completed in ten to fifteen minutes. As an incentive to completion entry to a free draw was organised with the prize of a free weekend holiday for two offered by the Friendly Hotel Group. In spite of this, and the inclusion of a stamped addressed envelope for the reply, the response rate (23 replies) was disappointingly low.

3. QUESTIONNAIRE RESULTS AND DISCUSSION

In spite of the disappointingly low response rate (23 out of 146), the questionnaires returned showed a wide range of software development activities, in terms of market, industrial sector, mode, design methods and programming languages and some interesting observations may be made. Unsurprisingly, most of the respondents were involved in the development of vertical applications - for specific clients or for inhouse use (effectively, specific clients). The data do not support the hypothesis that different "types" of software have specific profiles of desired quality attributes: e.g. 10 replies (over the full range of software types and modes).showed reliability, maintainability and usability (in any order) as the most important attributes. The response rate was probably too low to determine this type of pattern and the categories used for software were possibly not appropriate.

3.1 Quality Attributes

The main focus of the questionnaire was on quality attributes: their importance, achievement and measurement. The results (Table 3) show clearly that the attributes considered most important were reliability (much more important), usability and maintainability. All others were seen as much less important. The lack of importance for efficiency was encouraging. For some time lecturers have told students that greater processor power at lower cost and higher personnel costs are reducing its efficiency: this is born out by the results. We give below a summary of the reported means of achieving and measuring quality attributes. The number of respondents who seemed unable to distinguish between these activities was surprisingly high.

Rank / **Attribute**	*1*	*2*	*3*	*4*	*5*	*6*
Maintainability	2	6	5	8		
Usability	5	5	7	1		
Reusability		1	3	2	5	3
Portability	2		1		5	3
Reliability	14	5	2	1		
Efficiency		1	4	4	2	1
Others/unknown	2	2				

Table 3. Frequency of rankings of quality attributes

3.2 Achievement of Desired Quality

In general reliability seems to be achieved by the application of standard software engineering and project management methods and the responses seemed to reflect its perceived importance as a quality attribute. While two respondents stated that they used some form of formal method (e.g. Z) as a design method, neither of them mentioned this as a means of achieving reliability. By far the most commonly reported means of achieving reliability were walkthroughs/reviews (both design and code) and testing. Other methods mentioned once or twice included: Fagin reviews, static code analysis, use of a QMS (ISO9001), independent modules and modular design, use of testing tools.

While usability was also considered a very important attribute the means of achieving it were generally much less clear. The usual approach appears to be something like "see whether the customer/user likes it", often while showing the user screen designs. Prototyping was mentioned several times, but, while not totally clear, this usually seemed to either refer to screen designs or just showing the customer a reduced version of the final system. One response suggested a serious approach: "specially trained engineers using videoing and measurement". However, there was a serious lack of use of modern relatively formal approaches to interface design, such as task analysis and cognitive walkthroughs [12], which allow a considerable amount of interface design and evaluation without involving the user.

The means reported for achieving maintainability centred, fairly predictably, around the concept of "good practice". Specifically mentioned were: structured coding methods, modular design, standards (design and coding), reviews (design and code), common code.

The main means reported for achieving other attributes were:

portability:	use of appropriate coding standards
efficiency:	detailed analysis of task, design reviews
reusability:	programming standards, OOD/inheritance, reviews for common procedure identification.

One respondent stated that OOD had been a dismal failure with respect to reusability.

3.3 Measurement/Evaluation of Quality Attributes

The most commonly used approaches to measuring reliability were testing and fault logs. The latter were presumably based on faults reported by users. Other methods mentioned included: third party testing, defect seeding, MTBF, simulation, customer satisfaction. Quite a high level of sophistication was apparent in the methods used to evaluate reliability.

Methods reported for measuring usability ranged from "no formal assessment" to "time taken on user training" to "video, industry standard measures". Most responses were in terms of either user acceptance or customer satisfaction. As with the achievement of this attribute, there was very little attempt to produce any formal or rigorous measurement (for an overview see [13]).

Approaches to measuring maintainability were of two basic types. Indirect measures were based on the means of achievement: clear structured code, specifications and documentation, walkthroughs of module design. Direct measures were based on the effort expended on maintenance - either time or the amount of code needing to be changed: time taken to resolve problems and apply changes, speed of fault resolution, ease of bug fixing, proportion of modules changed. The indirect measures act as predictors of the direct measures. The measurement approaches were fairly sophisticated.

One of the two respondents who reported maintainability as the most important attribute gave as the method of assessment "don't normally assess"!

Respondents did not seem to have too much trouble measuring the other attributes:

portability:	3 supplier platform testing	time taken to port to other platforms
	amount/proportion of code rewritten	
efficiency:	processing/response time	benchmark tests
	speed of volume processing	user's productivity
reusability:	amount/percentage of code reusable	number of common routines
	use of Object Management System.	

One other attribute reported as being measured was defect density (defects per function point).

3.4 Design Methods

Encouragingly, all but three respondents reported using some sort of design method. Most common were prototyping (mentioned 11 times) and SSADM (mentioned 7 times). Those involved in real-time systems were the only users of Yourdon, OOD and formal methods, suggesting that such companies take design issues very seriously. The companies which seemed to take design least seriously also did little to assess quality attributes. The first choice design method (prototyping) was unexpected.

3.5 Other Observations

Several respondents reported using C++ as a programming language but did not use OOD as a design method. This suggests the possibility that the increasing use of C++ is as a "cleaner" version of C, rather than as a vehicle for implementing OOD.

One particularly detailed response reported the use of "incremental, organic development targeted at achieving greatest benefits early in project life": this seems close to the approach advocated by Gilb [4], described above. The same response mentioned a need for better software tools, particularly for the maintenance of old code. Others also desired configuration management and CASE tools.

4. CONCLUSIONS

The results of the questionnaire, while failing to confirm the original hypothesis, allowed some interesting observations. It is planned to continue this work, developing an improved questionnaire in the light of what has been learned. It is clear that other ways of characterising the type of software developed need to be found. An interesting comparison would have been between software developed for specific clients and for a general market. It is also felt that where examples (of languages, attributes etc) are given on the questionnaire, these should either be very few or comprehensive: in the case of the quality attributes listed on the questionnaire these were not comprehensive but the list was probably large enough to discourage respondents from adding their own.

REFERENCES

1. McCall, JA, Richards, PK, Walters, GF: 'Factors in Software Quality' *Vols I, II, III. US Rome Air Development Center Reports NTIS AD/A-049 014, 015, 055,* 1977
2. Boehm, BW, Brown, JR, Kaspar, JR, Lipow, M, MacCleod, GJ, Merrit, MJ: '*Characteristics of Software Quality'.* (North Holland, 1978)
3. Kitchenham, B: 'Towards a Constructive Quality Model. Part I: Software Quality Modelling', Measurement and Prediction' *Software Engineering J,* 1987, Vol.2, (4), pp105-113
4. Gilb, T: '*Principles of Software Engineering Management'.* (Addison Wesley, 1988)
5. Delen, GPAJ and Rijsenbrij, DBB: 'The Specification, Engineering, and Measurement of Information Systems Quality' *J of systems and Software,* 1992, Vol 17, (3), pp 205-217
6. Erikkson, I and McFadden, F: 'Quality Function Deployment: a Tool to Improve Software Quality' *Information and Software Technology,* 1993, Vol 35, (9), pp 491-498
7. Akao, Y: '*Quality Function Deployment: Integrating Customer Requirements into Product Design',* (Productivity Press, Cambridge, MA, 1990)
8. Garvin, DA: 'What Does "Product Quality" really Mean?' *Sloan Mgt Review,* 1984, Vol 25, Fall
9. Inglis, J: 'Standard Software Quality Metrics' *AT&T Technical J,* 1986, Vol 65, (2), pp 113-118
10. Daskalantonakis, MK: 'A Practical View of Software Measurement and Implementation Experiences Within Motorola' *IEEE Trans on Software Eng,* 1992, Vol 18, (11), pp 998-1010
11. Musson, T: 'Towards a Discipline of Software Quality' *Proceedings of the Third Software Quality Workshop, Edinburgh,* 1993
12. Polson, PG, Lewis, C, Rieman, J and Wharton, C: 'Cognitive Walkthroughs: a Method for Theory-based Evaluation of User Interfaces', *Int J Man-Machine Studies,* 1992, Vol 36, pp 741-773
13. Sweeney, M, Maguire, M and Shackel, B: 'Evaluating user-computer interaction: a framework', *Int J Man-Machine Studies,* 1993, Vol 38, pp 689-711.

Appendix A: The text of the questionnaire (omitting additional space left for replies):

Software Quality Profile

1. Is your company more concerned with producing: (please tick)
   a. General Purpose Software (horizontal) b. Specific Software Applications (vertical)
2. Is your software produced mainly for:
   a. In house use b. Specific customers
   c. A specific market sector d. General market
3. If possible give the "industrial sector" in which your software is principally used:
   (e.g. financial, engineering, education, leisure) . . . .
4. In which of the following modes is your software used:
   a. Real time b. Interactive c. Batch d. Networked e. Other (please specify)
5. What are the most important design methods and techniques (e.g. SSADM, object-oriented, prototyping) which you use in developing your software. (Please list in order of importance)
6. What types of computer language do you use to develop your software (e.g. 3GL, 4GL, database, assembly)? (Please give specific names and if possible indicate your % use of each language)
7. List the software tools you use in developing software (e.g. testbeds, software engineering environments). (Please give name and type of each tool)
8. Below is a list of quality attributes (attached sheet gives interpretation for each). Please indicate which are important to you, ranked in order of importance and if possible give weightings for their relative importance (100 = weight for most important)
   example: if the only attributes important to you are reliability (most important) and maintainability, then rank these 1 & 2 and do not rank any other attributes. If maintainability is only half as important to you as reliability then weight them as 100 & 50.
   (Please add and define any other attributes which are important to you)
   Maintainability Usability Reusability Portability Reliability Efficiency
9. Indicate briefly how you assess those quality attributes which you consider to be important (e.g. efficiency: time taken to process standard test data).
10. Explain briefly how you attain the appropriate quality level for each attribute that is important to you (e.g. reliability: design walkthroughs, static code testing, testbeds). (Please indicate how important you consider each means of achieving the desired level of quality)
11. Please indicate below any other issues which you feel are important to you in ensuring the quality of your software.