The Evaluation of Software Quality Factors in Very Large Information Systems

Souheil Khaddaj[1] and G Horgan[2]
[1]School of Computing and Information Systems, Kingston University, UK
[2]Nexis Associates Ltd, London, UK
s.khaddaj@kingston.ac.uk.

Abstract: A quality model links together and defines the various software metrics and measurement techniques that an organisation uses which when measured, the approach taken must be sufficiently general for hybrid hardware and software systems. In this work software quality factors that should be taken into account in very large information systems will be considered. Such systems will require a high degree of parallelism and will involve a large number of processing elements. We start by identifying the metrics and measurement approaches that can be used. Many of the quality factors would be applied in similar way for sequential and parallel/distributed architectures, however a number of factors will be investigated which are relevant to the parallel class. In such a system many elements can fail which can have major impact on the system's performance, and therefore it affects the cost/benefit factors. Portability and usability are other major problems that need to be taken into account when considering all the relevant factors that affect quality for such environments.

Keywords: Quality Modeling, Quality Measurement, Software Quality, Very Large Information Systems, Distributed Computing.

1. Introduction

There are a number of requirements that need to be met by a quality model, in order for confidence to be gained that the model correctly captures quality requirements, and correctly reflects how well those requirements have been met. A quality model links together and defines the various software metrics and measurement techniques that an organisation uses. The model answers the question "What is Quality?" and the management of the processes surrounding its use forms a Quality Assurance Process Management Programme, which is defined by the system of policies, procedures and guidelines established by an organization to achieve and maintain quality.

Achieving good performance in a very large information system requires a large amount of computing power, in terms of processors, memory and disk space. It requires the development of techniques for very large scale data handling, efficient strategies for physical clustering of and access to data on secondary storage, and globally distributed data management (Zhang et al, 2003) . This might involves advanced object-oriented modeling and design techniques, and it will require the use of emerging technologies such as computational grids in order to handle such a high degree of complexity (Czajkowski et al, 2001; Von Laszewski et al, 2002).

In the last few decades there have been substantial improvements in computer performance, due not only to advances in hardware but also to innovations in computer architectures, that is, how the computer is designed and organised to carry out its computational tasks. A major development that has affected the computer performance is distributed architectures, wherein the processors are replicated and organised such that they can act in unison on one application. The general acceptance of this technology has been slow for a number of reasons, including the wide differences of the available distributed architectures, which mean that there is no unifying software/hardware environment. It is also widely accepted that software/hardware development can be ad hoc and evolutionary. As a result, engineering environments may start off with poor quality. As the software development evolves so quality should improve.

In this work factors in software quality that should be taken into account when using a very large information system will be considered. We start by identifying the metrics and measurement approaches that can be used. A major example will be the lack of suitable performance metrics, which affect many quality factors such as the cost/benefit factor. Performance metrics for distributed software are tied to the target architecture, and there are as many of these as there are distributed architectures. Portability is another major problem; Changing computer architecture usually requires the rewriting of programs or readapting to a particular architecture and to a particular software environment. Usability is also important since it is harder and more time consuming to program

and use such systems when compared with sequential ones.

2. Quality modelling

Quality is a multidimensional construct reflected in a quality model, where each parameter in the model defines a quality dimension. Many of the early quality models have followed a hierarchical approach in which a set of factors that affect quality are defined, with little scope for expansion (Boehm et al, 1978). More recent models have been developed that follow a 'Define your own' approach (Fenton, 1991). Although an improvement, difficulties arise when comparing quality across projects, due to their tailored nature. The approach used in this work provides a framework in which global and locally defined quality criteria can be considered, individual quality views can be combined and view conflicts can be handled. Thus, it can be used as a standard of excellence measure for a Product, Process or Resource.

Considering that quality comprises implicit and/or explicit attributes, there are a number of views and opinions that need to be considered when defining and measuring quality. These views and opinions are referred to as Essential Views, and are determined by examining an individual's expected use of the product, their experiences in developing or using similar products. By concentrating on removing conflicts of opinion between the Essential Views, a consensus can be reached as to what properties constitute quality, and how quality should be measured.

The properties that constitute the 'explicit and / or implicit attributes' of quality form a set of Key Quality Factors and a set of Locally Defined Factors (Horgan, 2000; Horgan et al, 1999). The Key Quality Factors (KQFs) represent global quality criteria, i.e. factors that are required of all products, and their list of properties is static. The Locally Defined Factors (LDFs) represent local quality criteria, i.e. additional factors identified by the Essential Views, and are appropriate only to the current product being developed. In this way, the KQFs represent a common set of criteria that can be used for cross-project comparisons, whilst the LDFs retain the ability to allow local tailoring (figure 1). The LDFs are not a replacement for the KQFs. Instead, they define additional quality criteria. Their identification and inclusion is entirely the responsibility of the Essential Views. If the different views agree that additional criteria are required, then the additional criteria form the LDFs. In this work we only consider Key Quality Factors and for the rest of the paper they are referred to as simply Quality Factors.

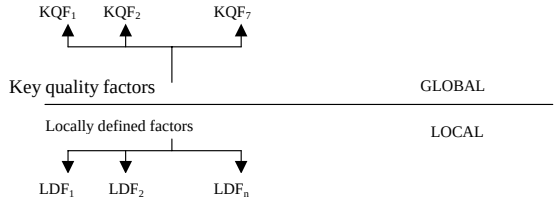

Figure 1: Global and local quality factors

2.1 Quality factors

Quality factors have been used in literature since the early hierarchical quality models (Boehm et al, 1978). The popularity of theses is reflected in the fact that the International Standard ISO 9126 is based on them. The standard recommends a number of factors such Reliability, usability, maintainability etc (Kitchenham, and Plfleeger, 1996). However, people tend to resist plans which evaluate many quality factors, due to limited resources or tight schedules. Based on previous research (Miyoshi and Azuma, 1993) the number of key factors should be kept between three and eight.

In this work a total, eight Quality Factors are defined. These are Performance, Scalability, Cost/Benefit, Usability, Portability, Robustness, Correctness, and Reliability. Many quality factors would be applied in similar way for sequential and distributed systems, as mentioned earlier we are mainly concerned with those specifics to distributed systems. Thus, many factors will not be included such as Maintainability, which is defined as the ability of a product to be modified, and Timeliness, which is defined as the ability of a product to meet delivery deadlines. If a product is delivered late, then it may have good quality aspects, but customers may rightly consider it to be of lesser quality than products delivered on time (Horgan, 2000).

Parallel/distributed processing offers the possibility of an increase in speed and memory beyond the technological limitations of single-processor systems. The performance of such systems can be varied and complex and users need to be able to understand and correct performance problems. Using distributed techniques it is possible to achieve higher throughput, efficiency, performance, and other advantages. Although parallel execution time, concurrency, scalability, and speed-up have been proposed in the literature as performance

metrics, the use of speed-up dominate the literature. The notion of speed-up was initially used to estimate the performance benefit of a multiprocessor over a uniprocessor (Kuck, 1978). Later, speed-up has been proposed as a metric to estimate the performance of parallel and distributed algorithms (Ghosh and Yu, 1998) and parallel processor architectures (Manwaring et al, 1994). The most common definition of speed-up is the ratio of the execution time of the best known sequential algorithm to the execution time of the parallel algorithm on a given number of concurrent processors. The execution time corresponding to a parallel execution is the longest of the CPU times required by any of the concurrent processors.

A general, commonsense definition of performance is presented in (Ferrari, 1978) wherein he defines performance as an indication of how well a system, already assumed to be correct, works. In general, the performance metric for distributed systems must reflect the program, the architecture, and the implementation strategies for it depends on each one of them. However, while the parallel execution time, speed-up, and efficiency serve as well known performance metrics, the key issue is the identification of the distributing processing overheads which sets a limit on the speed-up for a given architecture, problem size, and algorithm.

Every problem contains an upper bound on the number of processors, which can be meaningfully employed in its solution. Additional processors beyond this number will not improve solution time and can indeed be detrimental. This upper bound provides an idea as to how suitable a problem is for parallel implementation: a measure of its scalability. For scalability, we look for the ability to handle large numbers of processes and large or long-running programs on increasing number of processors.

Cost / Benefit is defined as the ability of a product to satisfy its cost/benefit specification. The Costs and Benefits involved in a product's creation should be a major consideration (Turnball, 1991). If the costs are high, and the benefits of its development are low, then there is little point in developing the product.

Usability is defined as the ability of a product to be used for the purpose chosen. It is a factor that is also considered important in other models (Gillies, 1992; Dromey, 1995). If a product isn't usable, then there is little point in its existence. To be useful, a tool should have adequate documentation and support, and should have an intuitive easy-to-use interface. On-line help is also helpful for usability. Adequate functionality should be provided to accomplish the intended task without putting undue burden on the user to carry out low-level tasks.

Because of the short lifespan of high performance computing platforms and because many applications are developed and run in a distributed heterogeneous environment, most parallel programmers will work on a number of platforms simultaneously or over time. Programmers are understandably reluctant to learn a new performance tool every time they move to a new platform. Thus, we consider portability to be an important feature.

For robustness, we expected the product to crash infrequently and its features to work correctly. Research tools are not expected to be as robust as commercial tools, but if the tool has been released for public use, considerable effort should still have been invested in debugging it and on error handling.

Correctness is defined as the ability of a product to meet and support its functional objectives. Other models also include this factor (Fenton, 1991). If software doesn't meet its objectives, then it may be reliable and it may be delivered on time, but no one will use it. Reliability is defined as the ability of a product to reproduce its function over a period of time, and is also included in other approaches (Kitchenham, 1987).

No other factors are included in the Quality Factors list. People tend to resist plans, which evaluate many quality factors, due to limited resources or tight schedules. Based on previous research, the number of key factors should be kept between three and eight (Miyoshi and Azuma, 1993). The Quality Factors set were chosen for their obvious importance for the particular system. However, it is accepted that only empirical validation across a large number of projects can determine the completeness of this set.

2.2 Relationship chart

The first step of the conflict removal mechanism is implemented by use of a Relationship Chart (Gillies, 1992). The chart displays graphically the relationships between quality criteria as a first stage towards measuring the criteria, and provides the basis for constraints on what can be achieved. In the

Relationship Chart, each criterion is listed horizontally and vertically. Where one criterion crosses another, the relationship between those criteria is specified. The relationships for the Quality Factors are fixed. Figure 2 shows the Relationship Chart for the discussed earlier Quality Factors.

By considering these relationships, checks can be made as to the feasibility of requirements. For example, users may state that a reliable product is required, that is both scalable and portable. The relationships between Reliability and Portability, and Portability and Scalability are set to Neutral. Therefore, it is acceptable to state a requirement for a reliable product that is also portable. Similarly, the relationship between Portability and Usability is set to Direct so it is also acceptable to state that a product be portable and usable. As a result, it is an acceptable requirement for a product to be reliable, usable and portable. Note that the Relationship Chart is only a tool and it is still necessary to check the detail of what is being asked.

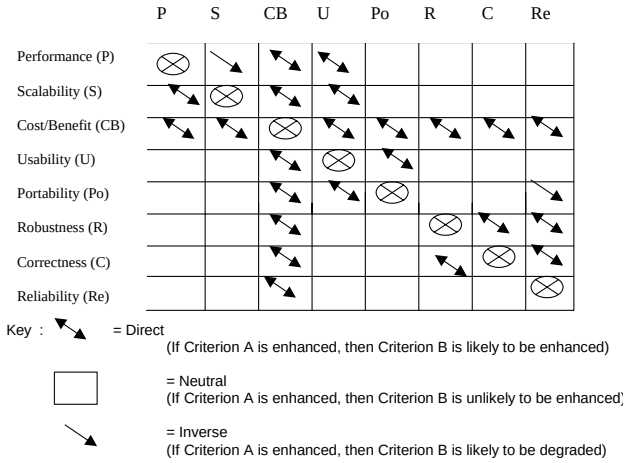

Figure 2: The relationship chart

2.3 Polarity profile

The second step in producing a consensus view of quality is to set the required goals for each criterion, based on the relationships identified in the Relationship Chart. In other approaches, a pie chart is used to represent quality goals (Pfleeger, 1993). There is a need to ensure that anyone can understand the graphical format chosen quickly and easily, particularly when it is considered that some essential views may belong to individuals with little technical background. There is also a need to illustrate over-engineered criteria (i.e., criteria that has exceeded its requirements), since further improvements in these areas will have little effect on the overall quality of the product. Such criteria cannot be shown easily using existing pie chart techniques.

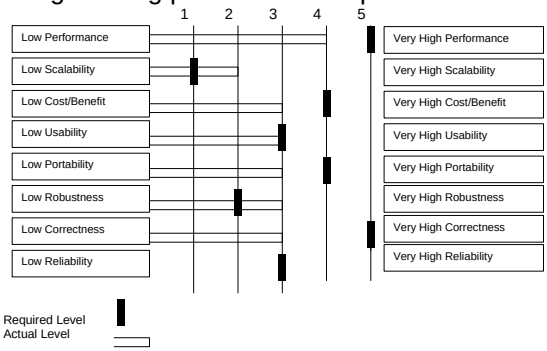

Figure 3: An example polarity profile

The solution chosen, therefore, is to use a Polarity Profile (Gillies, 1992). For each criterion, a range of values exists. The Required Quality of a criterion is defined as a single value on a horizontal line. The Actual Quality achieved is also defined as a single value on the same line. The advantage of using a Polarity Profile is that its format can be easily understood by anyone. Further, it is easy to determine whether or not a criterion has been over-engineered, since its Actual Quality value will be further advanced along the line than its Required Quality value. Figure 3 shows an example Polarity Profile. As can be seen, both Scalability and Robustness have been over-engineered, since their Actual Quality values exceed their Required Quality values. The criteria listed in the Polarity Profile are the same criteria as listed in the Relationship Chart.

Each organisation will use different metrics and metric approaches to measure different quality attributes. These metrics may be similar, identical or entirely different to those used by other organisations. In order to identify the Required Quality for each criterion in the Polarity Profile, the expected properties of that criterion need to be expressed using metrics. The same metrics should be used to identify the Actual Quality for that criterion. There is a need, therefore, for Conversion Mechanisms, which convert the results of metrics used to measure the quality of a criterion. However, for each criterion, the Conversion Mechanism will probably be unique to each metric used. Since different organisations may use different metrics, no single Conversion Mechanism will be suitable in all cases. The Conversion Mechanisms used, therefore, should be agreed between the Essential Views.

3. Conclusions

In the approach presented in this paper individual quality views can be combined, view

conflicts can be removed. It allows the specification of benchmarks against which achieved quality levels can be evaluated, and provides guidance for building quality into software for parallel systems. The feasibility of quality goals is controlled by the use of a Relationship Chart and a Polarity Profile. Since one cannot apply the constraint of having to define upfront the final requirements of a system (Butter, 1998), the approach is not static; if project personnel changes occur, or project requirements change, the Relationship Charts and Polarity Profiles can be updated to reflect these changes. Currently, a set of formal guidelines has yet to be finalised for identifying the Essential Views, despite their importance to the approach. For each occasion that the approach is used, time is required to identify the Essential Views and for those views to derive a consensus of opinions.

References

Boehm, B. W, Brown, J. R, Kaspar, H, Lipow, M, MacLeod, G and Merritt, M. J. (1978) *Characteristics of Software Quality*, North Holland.

Butter, M. (1998) 'Object Disorientation', *Business Computer World*, pp. 107-114.

Czajkowski, K., Fitzgerald, S., Foster, I., and Kesselman C. (2001) 'Grid Information Services for Distributed Resource Sharing' *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10),* IEEE Press.

Dromey, R. G. (1995) 'A Model for Software Product Quality', *IEEE Transactions on Software Engineering*, pp. 146-162.

Fenton, N. (1991) *Software Metrics : A Rigorous Approach*, Chapman and Hall.

Ferrari, D. (1978) *Computer Systems Performance Evaluation*, New Jersey: Prentice-Hall Inc., Englewood Clis.

Ferrari, C. (1997) 'The Road to Maturity : Navigating Between Craft and Science', *IEEE Software*, pp. 77-82.

Ghosh, S. and Yu M. L. (1998) 'An Asynchronous Distributed Approach for the Simulation and Verification of Behavior- Level Models on Parallel Processors', *IEEE Transactions on Parallel and Distributed Systems*, pp. 639-652.

Gillies, A. C. (1992) *Software Quality : Theory and Management*, Chapman and Hall.

Horgan, G. (2000) *Construction of a Quality Assurance and Measurement Framework for Software Project*, PhD Thesis, Kingston University, UK.

Horgan, G., Khaddaj, S. and Forte P. (1999) 'An Essential Views Model for Software Quality Assurance', *ESCOM-SCOPE 99*, pp. 387-396.

Kitchenham, B. (1987) 'Towards a constructive quality model', *Software Engineering Journal*, pp. 105-113.

Kitchenham, B and. Plfleeger S. L (1996) 'Software Quality: The Elusive Target', *IEEE Software*, pp. 12-21.

Kuck, D. J. (1978) *The Structure of Computers and Computations*, N.Y., John Wiley &Sons.

Manwaring, M. Chowdhury, M. and Malbasa V. (1994) 'An architecture for parallel interpretation: performance measurements', *Proceedings of the 20th EUROMICRO Conference*, UK, pp 531-537.

Miyoshi, T. and Azuma, M. (1993) 'An Empirical Study of Evaluating Software Development Environment Quality', *IEEE Transactions on Software Engineering*, pp. 425-435.

Pfleeger, S. L. (1993) 'Lessons Learned in Building a Corporate Metrics Program.', *IEEE Software*, pp. 67-74.

Turnball, P. D. (1991) 'Effective Investment in Information Infrastructures', *Information and Software Technology*, pp. 191-199.

Von Laszewski, G., Foster, I., Gawor, J., Schreiber, A and Pena C (2002) 'InfoGram: A Grid Service that Supports Both Information Queries and Job Execution', *Proceedings of the 11th IEEE International Symposium on High-Performance Distributed Computing*, IEEE Press, UK

Zhang, X , Freschl, J. and Schopf, J. (2003) 'A Performance Study of Monitoring and Information Services for Distributed Systems', *Proceedings of HPDC 2003*.

