ANALYZING, MEASURING & ASSESSING SOFTWARE QUALITY WITHIN A LOGIC-BASED GRAPHICAL FRAMEWORK

Nihal Kececi and Alain Abran

Department of Computer Science
Software Engineering Management Research Laboratory (SEMRL)
Université du Québec à Montréal
P.O. Box 8888, Centre-Ville Postal Station
Montréal (Québec) Canada H3C 3P8
nkececi@lrgl.uqam.ca abran.alain@uqam.ca

Résumé: La mesure de la qualité d'un logiciel doit supporter la gestion, le contrôle et l'amélioration du processus de développement du logiciel, mais les critères de qualité du logiciel ne sont toutefois ni bien définis, ni facilement mesurables. Cet article propose un cadre dynamique pour décrire la structure de systèmes d'évaluation capables d'englober les diverses dimensions (humains-logiciels-matériel) qui interagissent lorsque l'on évalue la qualité d'un logiciel faisant partie d'un système d'ensemble global de production. Ce cadre vise à faciliter l'identification des relations entre les caractéristiques de qualité du logiciel et du système global, des relations entre les requis de qualité et des caractéristiques mesurables, des mesure communes à plusieurs attributs de qualité et comment ces mesures peuvent être combinées pour une évaluation de la qualité du tout.

Abstract: Although software measurement is a key factor in managing, controlling and improving the software development process, software quality criteria are neither well defined nor easily measurable. This paper proposes a new logic-based graphical technique for modeling the dynamic interactions of the variables that affect software quality within a whole system production process. The framework presented here describes the properties of a complex quality assessment system composed of human-software-hardware interactions in terms of their quality requirements, and is designed to address the following issues: (1) What are the relationships between software and system measurable characteristics in terms of their contribution to whole-system quality? (2) What are the relationships between quality requirements and their measurable characteristics? (3) What are the common measures used to compute more than one quality attribute? (4) How can software-quality-related measures be combined to produce an overall assessment of quality?

Keywords: software quality model; modeling measurement complexity; dependability analysis; quality assessment for safety critical control systems.

1. INTRODUCTION

Although the term "quality" might seem self-explanatory in each application domain, in practice there are many different views of what we mean by quality, how it should be achieved and how it can be measured as part of a whole system production process. An increasing number of software quality standards for software processes and

products emphasize the need for measurement. However, most of these standards provide little guidance as to exactly what should be measured and how the results should be used in the assessment of software quality. Since the relative importance of the software quality acceptance criteria depends on the context of use and the purpose for which quality characteristics are being described, there is no general rule on how measures can be combined to produce an overall assessment of quality.

Software always runs as part of a larger system, typically consisting of other software products with which it is interfaced: human operators, hardware and workflow [ISO99]. Therefore, the whole-system characteristics have an influence on the criteria for software quality acceptance. This leads to issues such as: what the relationships are between the measurable characteristics of high-level software quality requirements, how they should be analyzed and how they can be made to work together to benefit high-quality integrated software systems.

To address these issues, we propose in this paper a conceptual quality assessment framework, developed using a graphical logic-based technique. Many of the ideas in this paper are derived from approaches developed in the system-engineering field, such as hierarchy theory and functional modeling. The flexible architecture of our quality assessment framework can be used for different sets of quality requirements in different application domains.

2. HIERARCHY THEORY AND FUNCTIONAL MODELING

System engineering views each system as an integrated whole, even though it is composed of diverse components, such as hardware, software and human subsystems. The objective of the system engineering discipline is to design subsystems which, when integrated into the whole, provide the most effective system possible to achieve the overall objectives. Some of the most challenging problems in building complex systems today arise in the interfaces between components. Hierarchy theory was developed to deal with the fundamental differences between one level of complexity and another.

Many modeling techniques have been developed and used in scientific disciplines such as artificial intelligence, risk assessment, reliability engineering and cognitive science. Each of these techniques is dedicated to a specific aspect of complex systems, and most utilize a functional/structural/behavioral modeling approach to describe a system.

Functional modeling is an approach used to model any man-made complex system by identifying the designer-defined overall goal it must achieve and the designer/user-defined functions it must perform. The characteristics and types of functional modeling can vary.

Functional modeling has been widely used for analyzing complex systems. The Goal Tree Success Tree (GTST), Dynamic Master Logic Diagram (DMLD) and GTST-DML methodologies are some examples of the implementation of this methodology for the analysis of complex systems [Mod&al 99], [Kec&al 99a], [Kec&al 99b].

3. SOFTWARE QUALITY AND RELATED VIEWS

Approaches to Measurement

Many authors have classified software-related measures as either product measures or process measures. Pressman [Pre97] classifies software measures along two orthogonal dimensions. Along one axis lie size-oriented, function-oriented and human-oriented measures. Technical,

quality and productivity measures occupy the other axis. Fenton [FEN97]] adds resource measures to these two. Resources are the items used in the creation of the software system, processes are the methods followed for creating software systems, and products are the end-results of process activities. Moeller & Paulish [MOE93] categorize software measures in terms of size, product quality and process quality.

Each software measure in any of the above categories can be either direct**,** meaning it can be directly measured from the entity itself, or indirect, meaning it is derived through transformation of some other measures.

Various researchers have produced models (usually taxonomies) to measure the software quality characteristics or attributes that to "rank" the level of achievement each of product's quality attributes. It can be useful for rating the quality of a software product. The models often include proposed ratios and formulae

Thus, each of the various types of empirical investigation plays a part in learning how various process, product and resource factors affect software quality.

Approaches to Software Quality Models

There are many software quality models that suggest ways to tie together different attributes. Each model helps us to understand how the several factors contribute to the whole. When we evaluate the quality of the product, we must see this big picture. Various researchers have built models to relate the user's external views to the developer's internal view of the software.

McCall's model of software quality incorporates 11 criteria encompassing product operation, product revision and product transition. McCall and his colleagues have shown how external quality factors are related to product quality criteria [McCall & al 77].
Boehm's model [Boehm & al 78] is similar to McCall's in that it presents a hierarchy of characteristics, each of which contributes to overall quality. His model is based on a wider range of characteristics and incorporates 19 criteria It has been noted that Boehm's notion of successful software includes characteristics of hardware performance that are missing in the McCall model [Pfleeger 98].

Dromey has addressed product quality by defining all the related sub characteristics in such a way that they can be measured and amalgamated into higher-level characteristics. [Dromey 96]

In the early 1990s, the International Standardization Organization ISO/IEC attempted to consolidate the many views of quality into one model. The ISO/IEC 9126 series standards have introduced a hierarchical model with six major quality characteristics, each very broad in nature. They are divided into 27 sub characteristics, which contribute to external quality, and 21 sub characteristics, which contribute to internal quality. ISO/IEC 9126-1 is concerned primarily with the definition of quality characteristics and sub characteristics in the final product. ISO/IEC 9126-2 gives examples for external quality metrics, the specified function of which is to measure such quality attributes as they relate to the operation and behavior of the system containing the software. ISO/IEC 9126-3 (under development at the time of this analysis) gives examples for internal quality metrics, the specified function of which is to measure such attributes as they relate to software quality characteristics.

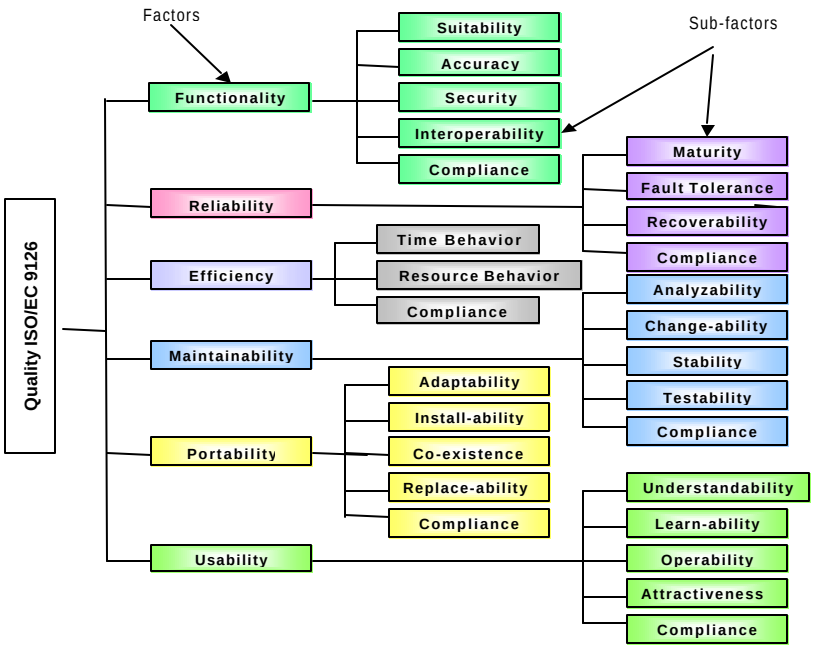

Figure 1 ISO/IEC 9126-quality model [1998]

Most of the external metrics use measurement values derived from test cases and the results of problem detection in validation testing or operation testing. Most of the internal metrics are also derived from review processes [ISO/IEC98].

Each model has a different set of attributes at the highest level of the taxonomy: selection of, and definitions for, the attributes at all levels may differ. They may also have a different number of hierarchical levels.
One major difficulty with these models is that the hierarchy is strict: each high-level quality characteristic (see Figure 1) is related to exactly a set of sub characteristic and/or only a set of quality attribute. High level quality factors are also assumed to be independent of each other, and are related to the user's view of the software, rather than to the whole-system (software-hardware-human) characteristics.

4. ADDRESSING THE PROBLEMS

These models, and many others, are helpful in articulating just what it is that we value in the software we build and use. But there are a number of difficulties in the direct application of any of the above models.

First, since quality has been defined as meeting requirements, it is not possible from a practical point of view to define one generic model, which can fit into all types of application domain. It is necessary, therefore, to develop a flexible conceptual quality framework. For instance, these models do not fully take into account either the requirements quality attributes (such as ambiguity, completeness, traceability, volatility, correctness, stability, etc.) or management measures (such as productivity, cost scheduling, etc.). Moreover, both dependability and integrity are quality attributes, which are recognized as critical properties in safety-critical systems, but

none of the above-mentioned models either includes them or discusses their potential relationships with whole-system quality.

Secondly, these models take for granted that all high-level quality attributes are independent of each other. Based on this assumption, they have decomposed higher-level quality factors into lower-level ones independently. While a high-level quality factor can be measured by a certain set of metrics, some of these metrics can be used for the quantification of other quality attributes. In addition, the data available to support an indirect measure could be used to support many other indirect measures. For instance, software size is a measure used to derive many indirect measures, such as defect density, effort, productivity, fault density, etc.

Thirdly, these models are static: they do not describe how to measure quality from current values at subsequent changes. Hardware and software can behave differently when changes are made. For instance, although repairs generally restore the hardware to its previous state, changes to a software requirement almost always change the software state. Measurement of system and software quality, therefore, should consider both deterministic and probabilistic approaches in the same measurement framework. In addition, the level of complexity between internal and external quality attributes has not been addressed yet.

Finally, the behavioral, functional and structural diversity of system components (such as software, hardware and human characteristics), as well as the interrelationships of measures, introduce additional complexity to existing approaches to software quantification. In summary, the complex relationships between direct measures and quality factors, as well as the complexity between indirect measures and quality factors, make it difficult to determine overall quality.

5. OUR APPROACH TO QUALITY ASSESSMENT

Building a Conceptual Framework for Analyzing Measurable Quality Variables

Every high-level software quality requirement can be a function of many variables of whole-system characteristics, and their behavior can change with use cases. When software quality requirements are defined for a specific application domain, the quality subfactors and/or attributes that contribute to the quality requirements should be identified hierarchically. To avoid confusion, we have used the terminology defined in IEEE 610.12 - quality factors, sub-factors and attributes - it is normally necessary to provide at least one measure for each of the attributes. For many cases, one measure can be used for the evaluation of more than one attribute, or vice-versa.

From these assumptions, the proposed quality assessment framework is built as a logic-based framework, with three major components: (1) Objectives, (2) functions and (3) primary data:

Objectives, including predefined quality requirements that can relate to human, hardware and software characteristics. Objectives-goals are decomposed until measurable characteristics - attributes can be identified.

Functions (indirect measures, models and/or base data), including prediction and estimation models. Functions (simple algorithms/ratios or complex formulae) are derived through the transformation of some other measures. For instance, simple functions – mostly ratios – are generally based on primary data that can be: interpreted. These data are used to obtain parameters calculated according to predefined formulae.

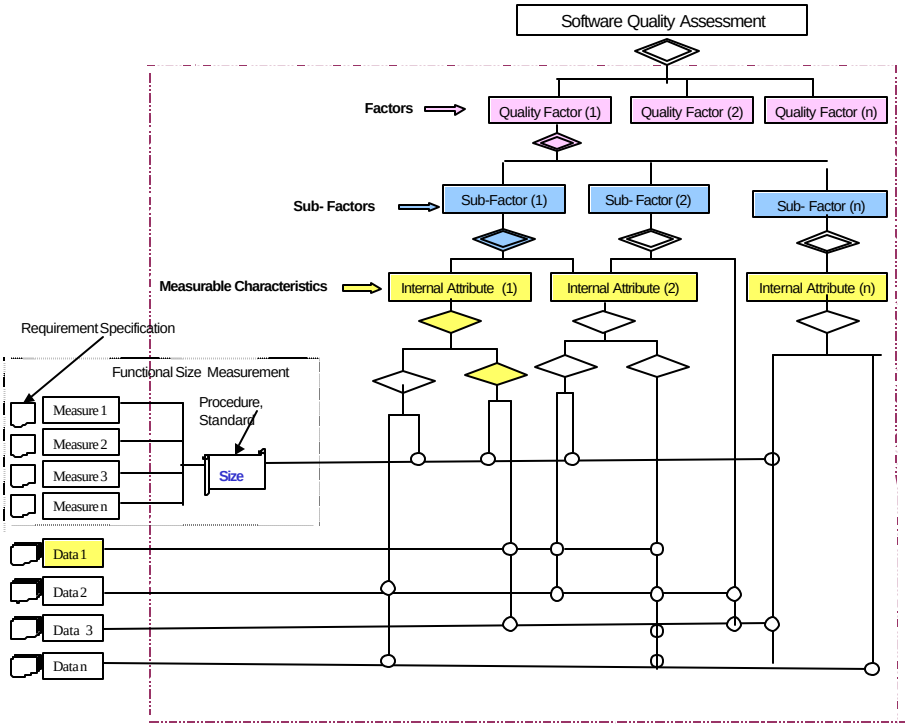

Figure 2 Graphical Dynamic Quality Assessment (GDQA) framework

Primary data: generally single values collected from process documentation or system/software specifications (e.g. number of errors, review effort in hours, etc.).

These components of a quality system interact with each other in a complex manner. For instance, while reliability is one major quality **objective**, reliability estimation models are **functions** of this objective. Maturity, fault tolerance, recoverability and compliance measures are **sub functions** influencing reliability with certain importance weightings. Fault density, which can be calculated with number of detected faults and product size, is an indirect measure of software reliability where the number of faults is a primary datum.

By contrast, reliability may be measured externally by observing the number of failures in a given period of execution time during a trial of the software system, and internally by inspecting the detailed specifications and source code to assess the level of fault tolerance.

6. GRAPHICAL DYNAMIC QUALITY ASSESSMENT (GDQA) FRAMEWORK

The totalities of software quality factors, which can be process-, product-, management- and/or human-related, are classified into a hierarchical tree structure, as illustrated in Figure 2. The highest level of this structure consists of quality factors and the lowest level consists of software quality attributes. Measurable characteristics of these factors are also a complex combination of whole-system characteristics as well as internal and external quality attributes. The main focus in the model is to specify the relationships between high-level quality factors and primary data (top to bottom). The framework can be used from bottom to top as well as from top to bottom.

The use of the proposed framework requires five steps:

1. The first step implements a structural hierarchy to decompose software quality factors into sub-factors and quality attributes. The decomposition process is repeated until some lowest level of measurable software characteristic is reached. In a multiple-layer hierarchy, the output of the first lower level can be directly linked to the inputs of the other layer sub-elements. The hierarchy may not be perfect, however, as some attributes may contribute more that one sub-factor and/or quality factor.

2. The second step assesses the priorities for the quality/subquality attributes: A weighting system will be used to make comparisons between attributes and to reflect the relative importance of distinct attributes. Weights are used to normalize with a sum to 1.0, for ease of comparison.

3. The third step defines the relationships between the factors and the sub-factors, since these sub factors, defined at the second or third level of the decomposition process, can have relationships across multiple quality attributes.

4. The fourth step identifies the indirect measures to quantify the values of software quality factors/sub-factors. This step provides a map from software attributes, which are decomposed into a multi-level structural hierarchy, to their associated available measurement techniques.

5. The fifth step determines the input variables that may be collected from the documentation of the development process (including requirements documentation, testing and maintenance reports and designs, as well as code).

7. CONCLUDING REMARKS

In this paper, we have proposed a new graphical dynamic quality assessment (GDQA) framework for modeling the dynamic interactions that affect software quality. This framework, and its components, is derived from system design engineering approaches aimed at making the entire modeling process objective, systematic and computationally fast. Many features and properties have been pointed out which make GDQA a flexible and potentially very useful alternative to current approaches in existing quality models. Furthermore, the GDQA framework helps facilitate:

(1) The identification of testing objectives;
(2) The use of a broad range of quality factors, sub factors, attributes, and their measures – direct/indirect, external/internal – for the whole system;
(3) The identification of the interrelationships between software-, hardware- and human-related characteristics that have an influence on the quality of the product;
(4) The identification, through its dynamic nature, of trends in quality by observing the time behavior of the variables;
(5) The identification of common measures used to compute more than one quality attribute.

REFERENCES

[Boehm & al 78] Boehm, B.W., J.R. Brown, J.R. Kaspar, M. Lipow, and G. MacCleod. *Characteristics of Software Quality. Amsterdam*: North Holland. 1978

[Dromey 96] Dromey, R.Geoff. "Cornering the chimera". *IEEE Software*, vol. 13, no 1, January, p. 33-34, 1996

[FEN97] Fenton, N. and S.L. Pfleeger. *Software Metrics: A Rigorous and*

Practical Approach, PWS Publishing Company, 20 Park Plaza, Boston, MA 02116-4324, 1997.

[Pre97] Pressman, R. *Software Engineering A Practitioner's Approach*, fourth edition, McGraw-Hill Companies, Inc., New York, 1997.

[**McCall & al 77]** McCall, J.A., P.K. Richards, and G.F. Walters. *Factors in Software Quality*, vol. 1,2, and 3, AD/A-049-014/015/055. Springfield, VA: National Technical Information Service, 1977

[Mod & al 99] Modarres, M., Y-S. Hu. *Applying Fuzzy-Logic-Based Hierarchy for Modeling Behaviors of Complex Dynamic Systems*. System & Software Computing in Nuclear Engineering. Da Ruan ed., Springer-Verlage (in Print), 1999.

[Kec&al 98] Kececi, N. and M. Modarres. "Software Development Life Cycle Model to Ensure Software Quality", Proceedings of the *4th International Conference on Probabilistic Safety Assessment and Management*, New York City, USA 1998.

[Kec&al 99a] Kececi, N., M. Modarres, and C. Smidts. "System Software Interface for Safety-Related Digital I&C Systems", *European Safety and Reliability – ESREL'99 Conference*, TUM Munich- Garching, September 13-17, 1999

[Kec&al 99b] Kececi N., M. Li, C. Smidts, C. "Function Point Analysis: An Application to a Nuclear Reactor Protection System," *International Topical Meeting on Probabilistic Safety Assessment –PSA'99*, Washington, DC, August 22-25, 1999.

[IEEE 90] IEEE Std. 610.12-1990. *IEEE Standards Glossary of Software Engineering Standards.*

[ISO/IEC 98] ISO/IEC 9126 *"Information Technology Software Quality Characteristics and Metrics"*; Part 1: Quality model, Part 2: External metrics, Part 3:Internal metrics.

[MOE93] Moeller, K.-H. and D. Paulish. *Software Metrics: A practitioner's Approach to Improved Software Development*. Chapman & Hall, and IEEE Computer Society Press, Los Alamitos, CA, 1993.