Proceedings of the

International Workshop on the Role of Software Architecture in Testing and Analysis

ROSATEA

Marsala, Sicily, Italy

Tuesday 30 June - Friday 3 July 1998

U.S. Program Chair:
Debra J. Richardson
University of California, Irvine
djr@ics.uci.edu

Italian Program Chair:
Paola Inverardi
Università di L'Aquila
inverard@univaq.it

Local Arrangements:
Antonia Bertolino
Istituto di Elaborazione della Informazione del CNR, Pisa
bertolino@iei.pi.cnr.it

Edited by

D. Richardson, P. Inverardi, A. Bertolino

i

Quality Evaluation based on Architecture Analysis

Fabrizio Fabbrini

+39 (0)50 593505
fabbrini@iei.pi.cnr.it

Mario Fusani
IEI-CNR
Via S. Maria, 46
56126 Pisa, Italy
+39 (0)50 593 512
fusani@iei.pi.cnr.it

Stefania Gnesi

+39 (0)50 593489
gnesi@iei.pi.cnr.it

ABSTRACT

A general, simple framework for quality evaluation is presented, in terms of entities (sets) and relations among them. Within this framework, a quality model to be used for architecture analysis is introduced, expressed as quality goals, architectural characteristics to achieve these goals, and activities to check for those characteristics. The model is a general one, so that existing methods and metrics can be included in it, as well as possible new devised methods and metrics. This can be useful for method comparison and selection.

Keywords

Quality models, quality evaluation, software architecture.

1 INTRODUCTION

Quality models have since long been introduced in literature (see for example [Deu88]), mostly as structured sets of properties (also known as *-ilities*, such as reliability, maintainability, and so on). These properties are usually presented as a hierarchy of statements, with a "expressed-in-terms-of" relation. The uppermost ones represent the most general properties, suitable to be understood by the widest variety of people, but very difficult to measure directly. The deeper we sink into this hierarchy, the more technical-audience-oriented and measurable properties are found. Conversely, links to upper levels get more uncertain and in most cases difficult to be defined. Terminological invention has played a great deal in this field, so, at various levels, properties are denoted as *goals* or *attributes* or *characteristics*, down to *sub-characteristics* or *factors*, to *criteria* and *indicators* and attributes again; but the point is not yet set, even in standardisation activities which are now covering the field [ISO91]. In the following we use some of these terms, in what we believe a self-explaining way, but with no wish to fix a glossary ourselves.

A quality model is normally used as a reference for evaluating to which extent an object possesses some expected properties, defined in the model itself. Some form of quality model, explicit or not, always is adopted when an evaluation is performed. Most often the model is implicitly assumed, or is *ad hoc* defined for the particular evaluation. Quality models also are used as references when designing and implementing objects, but even in this case an evaluation, typically performed step-by-step, occurs.

The purpose of this paper is twofold: 1) to define a quality model out of a very general framework, and not as a sort of unjustified "good thinking"; 2) to apply the model to architectural design evaluation, being this a nice stage in the software lifecycle to check for quality, by showing that various evaluation criteria and methods fall within the model. Then, no specific method is proposed in the paper, but rather a way is given towards method comparison.

In Section 2 the general quality framework and the model for quality evaluation based on architectural evidence are presented. In Section 3 existing evaluation methods are shown, along with their fitting into the model.

2 CONCEPTUAL FRAMEWORK

The concept of quality cannot be given as a simple definition and perhaps it is better not to define it. As quality involves human beings, any ultimate definition would likely find little application. Structured sets of properties, as mentioned in the Introduction, could be viewed as a definition, or, better, a *predicate* about quality.

Our approach, instead, is to introduce some basic entities dealing with the (undefinable) quality concept, and express our model as relations among these entities. Some bibliographical research work led us to find in the literature the presentation of a conceptual framework devised by Krogstie et al. [Kro95], that we have adapted to our purposes.

As quality evaluation is best effective if done as early as possible in the lifecycle (to allow for precocious quality flaw detection), the model is centered on software architecture description. It is important to point out that the quality to be evaluated regards the final software product (people buy products, not designs): architecture design is just the information source for such an evaluation. However, there is such a concept as quality of architecture, and this will be mentioned when needed.

The conceptual framework and model is shown in Figure 1. The entities involved (the rectangles) are sets of expressions, defined as follows:
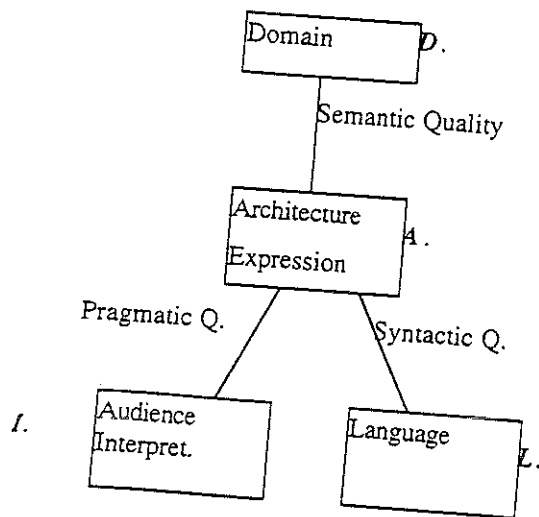
Figure 1 – Conceptual framework for quality

L (language) = all the expressions in any possible architectural description language, including structural relations and syntactic rules;

A (architecture) = all the expressions contained in the actual architectural documents;

D (ideal domain) = all the correct and relevant expressions for the actual architectural design, in all the languages L_i of L;

I (interpretation) = $I_1 \cup I_2 \cup \ldots \cup I_k$, all the various expressions of A as understood by audience (see below) individuals or groups.

Now some further explanation is needed about the sets D, L, I.

D is not, and generally cannot, be completely known. The purpose of introducing D is not for reference or checking, but to work out a rather rigorous definition for semantic quality (see below). Actually, some knowledge of D may be expressed by adequately refining functional requirements into architectural expressions, or, depending on the chosen architectural language, by mapping correctly these requirements into an object-oriented scheme. D also includes:

- descriptions of all the possible impacts of non-functional requirements (as performance and quality requirements) on the architecture;

- descriptions of all the requirements which directly address the architecture, such as modularity, module redundancy for fault tolerance, constraints for adopting COTS or reuse, constraints for distributed architecture.

Expressions in D are composed according to rules defined in the languages of L, and, conversely, L includes all the languages (natural, graphic, formal) which are appropriate for D. L also provides appropriate languages for A and I.

The audience (not directly included in the framework: all the sets, as mentioned, are homogeneous, that is composed of linguistic expressions) is made of all the entities (persons, organisational units, computer programs, etc.) which are supposed to get and possibly to share some interpretation of A:. These include designer, supplier, customer, evaluator, static analyser, translator, etc.

2.1 Quality Model

Figure 1 also shows our quality model expressed as a sort of matching relations (the links) between the set A and its neighbours D, L, I.

Quality comes out as a collection of desired properties, or goals. More precisely, three *quality types* are defined in this model, each giving rise to one or more *quality goals*. Here a hierarchical structure as mentioned in the Introduction appears, but in a more deductive way than those presented in literature:

Quality type: *Semantic quality*. It deals with the correspondence between A and D. Two quality goals are defined:

- *Semantic validity*, obtained when:

$$A_i \subseteq D$$

for any language L_i of L;

- *Semantic completeness*, obtained when:

$$A_i \supseteq D_i$$

for all the languages to be used in A..

The implicitly assumed language-dependent partition of A and D is obvious. What is not obvious is how to check for semantic quality: This cannot be deduced from the above definition. This point is touched in the next section.

Quality type: *Syntactic quality*. We have one quality goal:

- *Syntactic validity*, obtained when:

$$A \subseteq L.$$

Quality type: *Pragmatic quality*. Semantic validity is not strictly a pre-requisite, but it makes sound the following definition. Two quality goals are defined:

- *Valid comprehension*, obtained when:

$$I_i \subseteq A$$

for any individual in the audience, independently of the language;

...national Workshop on The Role of Software Architecture in Testing and Analysis
...ci. Sicily, Italy • 30 June - 3 July 1998

47

- *Complete comprehension*, obtained when:

$$I = I_1 \cup I_2 \cup \ldots \cup I_k \supseteq A$$

independently of the language. In practice, the language could play an important role with interpretation: Translations into equivalent architectural expressions may be needed to achieve complete comprehension (in this case the above formula would be slightly modified).

As already pointed out, the quality goals presented here are difficult to achieve, and generally impossible to be totally verified. However, we think we have somewhat broadened the traditional view of quality and, starting from the quality classification presented above, we can further extend the model to include modeling elements

that allow to relate various techniques for architecture analysis with the appropriate quality types.

3 EXTENDED QUALITY MODEL

By analysing the quality goals it is possible to find for each of them a set of sub-goals, or criteria, to be met for their achievement. In order to satisfy these criteria, some activities, in turn, can be put in practice.

In Table I a description of the quality model is given, extended to include some criteria and activities.

Both criteria and activities are to be further analysed and expanded. Activities should also be complemented by measurement procedures and metrics.

Quality types	Quality goals	Criteria	Activities/techniques
Syntactic	Syntactic validity	Lexicon correctness	Lexicon checking
		Graphical element selection	Graphical element checking
		Syntactic correctness	Syntax checking
		Structural correctness	Structure checking
Semantic	Semantic validity	Functional requirements compliance	Inspection, traceability analysis, data-flow analysis, O-O / requirements mapping analysis
		Quality requirements compliance	ISO/IEC-9126 inspection, complexity analysis, coupling & cohesion analysis, element counting
		Specific architectural requirements compliance	Inspection, element counting
	Semantic completeness	Domain-dependent completeness	Traceability analysis
		Domain-independent completeness	Traceability analysis, consistency checking
Pragmatic	Valid comprehension	Readability	Complexity analysis, coupling & cohesion analysis, readability formulae, language translation, joint reviews
	Complete comprehension	Navigability	Browsing, joint reviews

Table I – Extended quality model

REFERENCES

[Deu88] Deutsch M. S., Willis R. R. (1988) *Software Quality Engineering*, Randall W. Jensen.

[ISO91] ISO/IEC 9126: 1988, 1991 (IS), Information technology - Software product evaluation - Quality characteristics and guidelines for their use

[Kro95] J. Krogstie, O. I. Lindland, and G. Sindre, Towards a deeper understanding of quality in requirements engineering, *7th International CAiSE Conference, vol. 932 of Lecture Notes in Computer Science*, pages 82-95, 1995

International Workshop on The Role of Software Architecture in Testing and Analysis
Marsala, Sicily, Italy • 30 June - 3 July 1998

48