International Journal of Information Technology and Knowledge Management
January June 2009, Volume 2, No. 1, pp. 131-134

METRICS-BASED EVALUATION OF QUALITY OF NON-FUNCTIONAL SPECIFICATIONS

Simrandeep Singh Thapar[*], Hardeep Singh[**], & Karanjeet Singh Kahlon[**]

Non-functional Specifications or 'ilities' are critically important at the time of selection of components when there are already many candidate components providing functional capabilities. Quality aspect of non-functional metrics is very important in selection procedure of components. With the help of metric it becomes easier to identify the best suited component. In this paper we evaluated quality of non-functional specifications which is based on metrics.

Keywords: Quality, availability, reliability, maintainability, metrics

1. Introduction

Software companies in development of software products come across with basic constraints like Time to market and less cost. These can be reduced to some extent by using components. Components should be used only after cost-benefit analysis. Component based development (CBD) is in great demand for the development of software products from the components. Products are no longer developed from scratch; they are instead an assembly of components developed independently of the products [Roger S. Pressman 2001].Components are the software units which act in composition or independently [Szypersky 2002]. A component conforms to and provides the physical realization of a set of interfaces. Using components offers many advantages: *Reuse* to increase productivity and quality that further effect positively on performance and reliability, functionality is instantly accessible to the developer, cost cutting to some extent and time to market. As it is known, nothing is perfect, it applies to the components as well, as often, only a brief description of its functionality is provided with a component, component carries no guarantee of adequate testing, only a limited description of the quality of the component, developer does not have access to the source code of the component. Component are developed using frameworks like CORBA, JavaBeans, COM, .NET etc.

Component specification aims to provide a basis for the development, management and use of components. A Component Specification makes it easier to buy, sell, and replace components. Component-based systems require a renewed emphasis on specification and verification, because if one is to build a computer system based on components built by others, then one must know what each component is supposed to do and trust it to carry out that task. Similarly, the builder of a component needs to know what behavior its users depend on, so that improvements in algorithms and data structures can be made. A specification of a component can meet both these needs, since it acts as a contract between builders and their clients. Software component research community emphasized more on functional details of components but without non-functional details it would be quite cumbersome to identify or to make decision to select components out of components that provide all the functionality. So, Non-functional specifications prove decisive at the time of selection. Further, selection of particular component depends on functionality, quality, confidence in manufacturer and satisfaction of some constraints etc. Except functionality, all are non-functional or extra-functional properties. Specification of a component must consists of: A precise definition of the component's operations, all context dependencies (how and where the component can be deployed) along with non-functional properties or quality attributes which always prove decisive when component selection is to be made. There are several specification frameworks which are used in academia and the industry, for instance Eiffel, UML/OCL, VDM, B, Z, Larch/LSL, SDL, and RAISE/RSL [Uwe Keller 2005].

This paper has following sections after introduction, section 2 discusses further non-functional aspects with focus on quality aspects, section 3 discusses metrics approach to evaluate quality, section 4 concentrates on availability aspect of non-functional specifications and section 5 concludes this research paper.

2. Quality Perspective of Non-Functional Properties

The role of non-functional (or qualitative) specifications becomes more important because ready-made software components have their functionality already built-in. Non-functional requirements often correspond to strategic or

[*] Sr. Lect., Department of PG Studies, ACET, Amritsar. *E-mail: simthain@yahoo.com*

[**] Professor, DCSE, Guru Nanak Dev University, Amritsar. *E-mail: hardeep_gndu@rediffmail.com, karankahlon@yahoo.com*

business objectives of end-user organization as a whole, and, therefore, are likely to have higher priority if conflicting with some of the functional requirements for the software component. [Ljerka Beus-Dukic 2000] Non-functional properties are defined as the restrictions on the software product and are associated with the QoS which encompass "ility" properties for e.g. maintainability, usability, portability etc. Fig.1 shows that a component presents interfaces to customers to assemble the component in software and each interface can provide many operations. Here, credential is a combination of Attribute, Value and Credibility, where Attribute is a description of a property of a component, Value is a measure of that property and credibility is a description of how the measure has been obtained.

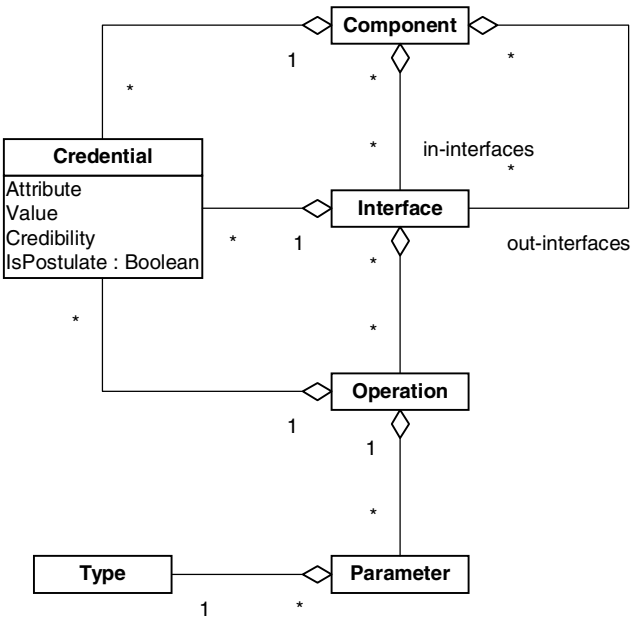

Figure 1: Non-Functional Specifications

In [L. Beus-Dukic 2000] explains three types of non-functional specifications for components. *Architecture specifications ad*dress a component's ability to be integrated into a system; these include performance characteristics, reliability, security, reusability and portability, *Domain requirements* describe properties related to the component's environment and *Organizational requirements* focus on the aspects of the vendor and customer. We focused on the evaluation of architecture properties especially quality.

According to the [IEEE 610.12] standard, as an attribute for a software system, software quality is

(1) The degree to which a system, component, or process meets specified requirements.

(2) The degree to which a system, component, or process meets customer or user needs or expectations.

After reviewing the literature, it is found that there is no consensus on how to define and categorize software quality characteristics.

3. METRICS-BASED QUALITY EVALUATION

A model is required for quality evaluation and evaluation can be qualitative or quantitative. A typical example of qualitative evaluation is an expert's opinion on the component artifact. This evaluation is subjective, and as known poses problems in comparison and generalization. The quantitative approach to evaluation provides, a more pragmatic way of dealing with this problem. It consists of defining, collecting and analyzing objective quantitative metrics that can be combined into a quality model [Miguel Goulão, Fernando Brito e Abreu 2004]. We will take up only those properties which deal with quality and try to follow as much as possible a standard terminology, in particular the most apt quality model ISO 9126. Our research uses the metrics to quantify the concepts of quality. A quality characteristic is a set of properties of a software product by which its quality can be described and evaluated. A characteristic may be refined into multiple levels of sub-characteristics [Manuel F. Bertoa, 2002].

ISO 9126 Model for Software Components

Characteristic	Sub-characteristic (Runtime)	Sub-characteristic (Life cycle)
Functionality	Accuracy Security	Suitability Interoperability Compliance Compatibility
Reliability	Recoverability	Maturity
Usability		Learnabilitiy Understand-ability Operability Complexity
Efficiency	Time behavior Resource behavior	
Maintainability		Changeability Testability

A metric can be assigned to the quality characteristics, where a metric is a procedure for examining a component to produce a single datum, either a symbol (e.g. Excellent, Yes, No) or a number. Metrics that are used to measure are: *ratio* specifies the number in percentage, *presence* specifies the existence whether present or not, *time* specifies the time duration and *integer* specifies the number.

Quality Attributes for Components Which are Measurable at Runtime

Sub-characteristics	Attribute	Type
Accuracy	Precision	Ratio
Security	Data Encryption	Presence

By having a framework fosters a more objective analysis of quality properties, partially mitigating the shortcomings of narrative reviews. A metric framework is proposed by [Miguel Goulão 2004].

- **Scope** – granularity level
- **Intent** – main objectives
- **Technique** – how the metrics are defined and validated
- **Critique** – a qualitative assessment of the noticeable features of the proposal
- **Maturity** – the maturity level

The first four items of this structure aim to provide a very brief overview of the proposals, while the last aims to characterize each proposal according to its maturity level. To assess the maturity, we start by identifying a set of rating scales concerning different aspects of metrics based quality evaluation. For each of those rating scales, we then identify several levels of maturity that will aid us in the graphical depiction of proposals maturity. Table 1 presents a condensed view of our maturity comparison framework.

Table 1
A metrics Proposal Maturity Comparison Framework

Maturity level	*Quality Mode (QM)*	*Mapping Quality (MQ)*	*Metrics definition (MD)*	*Level of Validation (LV)*
0	N/A	N/A	N/A	N/A
1	Ad-hoc	Ad-hoc	Wish list	Anecdotal
2	Structured	Rationale	Informal	Small experiment
3	Uncorrelated	Goal-driven	Semi-formal	Industrial experiment
4	Validated	Validated	Formal	Independent

4. EVALUATION OF AVAILABILITY CHARACTERISTIC OF QUALITY

In this paper, we are presenting a set of measures to assess the Usability of software components. First of all *Availability* needs to be defined. It is continuity of service to the customer. A component is available if it is in operational state and not down for repairs or maintenance [K.Shridhara Bhat 2007].

Availability = uptime / (uptime+ downtime)

Further, here we need to specify *Reliability* which refers to the length of time that a component can be used before it fails. In other words, reliability is the probability that a component will function for a specified period of time without failure.

The reliability of the component is also related to meantime between failures (MTBF) which is just the average time that the component functions from one failure to the next. The longer the MTBF, the more reliable the component.

Mai*ntainability* refers to the restoration of a component or service once it has failed. Since all customers consider maintenance or repairs as a nuisance, a high degree of maintainability is desired so that the product can restored to be used quickly. Maintainability can be measured by the mean time to repair (MTTR) the component.

Availability then is a combination of reliability and maintainability. If a component is high in both reliability and maintainability, it will also be high in availability. Availability then can be expressed as foll*ows*:

Availability = MTBF/MTBF + MTTR

For example, if a component has an MTBF of 8 hours and a MTTR of 2 hours each time it fails, then its availability is calculated as follows:

Availability = 8/8 + 2 = 0.8 or 80%

5. CONCLUSION

In specifying non-functional properties difficulty is clear. There are some common shortcomings of current approaches like ambiguity in definition, lack of adequacy of specifying formalism and insufficient validation of current quality models and metrics for software components. Szyperski suggests that the big-O complexity of a component be specified to indicate its time and space costs. [Sitaraman 2001] outlines a number of shortcomings of this solution. Although there are many difficulties specifying the non-functional properties and requirements of components they are essential to users during the selection and evaluation of components. Often, the non-functional properties set components with similar functionality apart.

References

[1] Pressman, Roger S., "Software Engineering: A Practitioner's Approach", McGraw Hill India (2001).

[2] Szyperski, C., Gruntz, D. and Murer, S. "*Component Software: Beyond Object-Oriented Programming"*. New York, ACM Press - Addison Wesley (2002).

[3] Uwe Keller, Jos de Bruijn, "Functional Specification in Common Specification Frameworks for Software Components", *WSML Working Draft* D28.1 v0.1 January 18, 2005 *http://www.wsmo.org/2005/d28/d28.1/v0.1/20050118/*

[4] Ljerka Beus-Dukic, "Non-Functional Requirements for COTS Software Components", *http://users.wmin.ac.uk/~beusdul/papers/cotsw00.pdf*

[5] IEEE Standard Glossary of Software Engineering Terminology, *IEEE Std* 610.12-1990, (10 Dec. 1990)

[6] Miguel Goulão, Fernando Brito e Abreu, "*Software Components Evaluation: an Overv*iew", CAPSI2004.

[7] Manuel F. Bertoa, "*Quality Attributes for COTS Components*", *Proceedings of the 6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE2002)* **1**(2), (2002), 128–144.

[8] K. Shridhara Bhat, "*Total Quality Management*", Himalya Publishing House, 1st Ed. (2007)

[9] M. Sitaraman. "Compositional Performance Reasoning". I*n 4th ICSE Workshop on CBSE: Component Certification and Systems Prediction (ICSE 23),* Toronto, ON, (May 2001).