J. Software Engineering & Applications, 2010, 3, 556-560
doi:10.4236/jsea.2010.36064 Published Online June 2010 (http://www.SciRP.org/journal/jsea)

On Some Quality Issues of Component Selection in CBSD

Jeetendra Pande[1], Raj Kishore Bisht[1], Durgesh Pant[2], Vinay Kumar Pathak[3]

[1]Department of Computer Science, Amrapali Institute of Technology & Sciences, Haldwani, India; [2]Department of Computer Science, Kumaun University, Uttarakhand, India; [3]Uttarakhand Open University, Haldwani, India.
Email: jeetendrapande@yahoo.com

Received March 11th, 2010; revised March 27th, 2010; accepted March 29th, 2010.

ABSTRACT

Component based development offers many potential benefits, viz. software reuse, reduced time-to-market, interoperability, ease of quality certification etc. However, it is not always that benefits derived from addition of components from a component repository are more than the costs involved in developing the module from scratch. This work evaluates various software quality models and suggests recommendations for enhancing software quality in COTS (*component off-the-shelf*) *based software products by designing software quality metrics that would help in managing and enhancing quality in component-based software development.*

Keywords: *Component Based Software Development* (*CBSD*), *Components-off-the-Shelf* (*COTS*), *McCall's Model, Dormey's Model, Bohem's Model*

1. Introduction

Software is at the heart of most industrial systems in use today. Rapid changes in industrial methods have led to a situation where industrial products are now, more often than not, systems consisting of software and hardware. Industries in which the use of software is now essential include, among others, Automotive, Medical-Systems, Process-Control and Manufacturing. In these and others, value added to products is provided largely by the associated software.

The economics of software projects has been seen as an important sub-discipline of software engineering for some time. Projects need to be considered in a wider business environment in which the issues of cost, schedule and productivity are of considerable significance. A key area in dealing with cost and quality of software systems is the ease and benefits accruing from reuse, *i.e.* how much can be saved by using existing software components when developing new software systems? To this end, Component Based Development (CBD) is widely accepted as the approach that could best yield the long sought after benefits of software reuse, reduced time-to-market, interoperability and ease of quality certification. The basis for reuse is the reliability of the components intended for reuse and the gains achieved through their application.

It is important to demonstrate to management and funding agencies that reuse makes good business sense. For this, it is necessary to have methods to collate and furnish clear financial evidence of benefits of reuse in real projects. To achieve this, we need to define good metrics that capture these benefits and develop tools and processes to allow effective use of these metrics. Deployment of component-based applications is a common practice in the area of commercial software. To meet market requirements, new functionality is frequently added to industrial products. Introduction of new functionality is often achieved by adding components to an existing system. This is why component-based development approach is attractive to industry.

Cost and time-to-market issues are addressed by taking recourse to the rapidly emerging component-based development (CBD) approach. Adding new functionality to existing products must result in lower cost and higher quality. In CBD, software systems are built as an assembly of components already developed and prepared for integration. The many advantages of CBD approach include effective management of complexity, reduced time -to-market, increased productivity, and greater degree of consistency and a wider range of usability. Thus, quality of the final software is highly influenced by use of CBD approach. Each component will have its own quality attribute profile, but when interfaced and used together with other components, the resulting composition may show a different quality attribute profile altogether [1]. A

large range of components, which perform the same function, are available from different vendors. This makes it very difficulty for a developer to decide which component to use and which to discard, based on the quality attributes of available competing components. Quality of an individual component is important but there is no guarantee that integration of components with high quality attributes will lead to a software product with overall high quality attributes. When multiple components are integrated, it is very difficult to reason about the overall quality of the final product and developers require some metric that helps them in evaluating and choosing components in such a manner that the final product is of high quality.

Project managers usually lay stress on the importance of improving estimation accuracy and techniques to support better estimates. This paper surveys the status of quality framework in component based software development and provides recommendations for future work in improving the quality of component based software development.

2. Software Quality Models

According to IEEE 610.12 standard [2], software quality is:

- The degree to which a system, component or process meets specified requirements.
- The degree to which a system, component or process meets customer or user needs or expectation.

The following section briefly explains the various quality models for software development.

2.1 Mccall's Model (1977)

The first quality model was proposed by Jim McCall *et al* [3,4], in 1977. This model is also known as the General Electric Model. McCall split eleven different quality attributes into three groups, as shown in **Table 1**.

1) Product Operation: It includes correctness, reliability, efficiency, integrity and usability.

2) Product Revision: It includes maintainability, flexibility and testability.

3) Product Transition: It includes portability, reusability and interoperability.

With the aim of improving quality of software products, McCall included a number of quality factors in his model that gave due weight to both users' views and developers' priorities.

Table 1. McCall's model

Product Operation	Product Revision	Product Transition
• Correctness		
• Reliability	• Maintainability	• Portability
• Efficiency	• Flexibility	• Reusability
• Integrity	• Testability	• Interoperability
• Usability		

The advantage of McCall's Quality Model is the relationship created between quality characteristics and metrics. The major disadvantage is that functionality of the software product is not considered.

2.2 Boehm's Model (1978)

Barry W. Bohem gave the second quality model in 1978. Boehm's model is similar to the McCall's Quality Model in that it also presents a hierarchical quality model structured around high-level characteristics, intermediate-level characteristics and primitive characteristics – each of which contributes to the overall quality level [5,6].

High-level characteristics represent high-level requirements of actual use. These characteristics address three main questions:

1) As-is utility
2) Maintainability
3) Portability

These three primary uses had quality factors associated with them, which represented the next level of Boehm's hierarchical model, the intermediate-level characteristics. Boehm identified seven quality factors, namely:
(a) Portability (b) Reliability (c) Efficiency (d) Usability (e) Testability (f) Understandability (g) Flexibility

The lowest level structure of the characteristics hierarchy in Boehm's model is the primitive characteristics metrics hierarchy, in which these quality factors are further broken down into *Primitive constructs* that can be measured. For example, *Testability* is broken down into accessibility, communicativeness, structure and self-descriptiveness. Bohem's model lays emphasis on maintainability of the software product. It includes considerations involved in evolution of a software product with respect to the utility of the program. Boehm's quality model is based on a wide range of characteristics but does not elaborate on the methodology of measuring these characteristics.

2.3 FURPS/FURPS+ (1992)

This model, proposed by Robert Grady and Hewlett-Packard Co, decomposes characteristics into two different categories of requirements:

- Functional Requirements (F): Defined by input and expected output.
- Non-functional Requirements (NF): Usability, Reliability, Performance and Supportability.

FURPS is an acronym representing a model for classifying software quality attributes (requirements):

- Functionality–Feature set, Capabilities, Generality, and Security.
- Usability–Human factors, Aesthetics, Consistency and Documentation.

- Reliability–Frequency/severity of failure, recoverability, predictability, accuracy and mean time to failure.
- Performance–Speed, efficiency, resource consumption, throughput and response time.
- Supportability–testability, extensibility, adaptability, maintainability, compatibility, configurability, serviceability, installability, localizability and portability.

The main disadvantage of the FURPS model is it does not take into account the portability of the software product. Rational Software extended this model to FURPUS+, for use in corporate requirements such as design constrains, implementation requirements, interface requirements and physical requirements.

2.4 Dromey's Model (1996)

R. Goeff Dromey proposed this model in 1996 [7,8]. According to Dormey, high level attributes like reliability and maintainability cannot be built into software. Therefore, he identified a set of properties to cover these requirements. In this model, Dormey focused on those properties of the software product that affect the quality attributes. He connected software product quality with software quality attributes (**Table 2**).

The disadvantage of this model is that efficiency of software is not considered for determining the quality of software.

2.5 ISO Model 9126 (1991)

ISO 9126 model was proposed in 1991. Unlike its predecessors, MaCall's Model and Bohem's Model, upon which this model was built, functionality of a software product is also taken into consideration in this model. It proposes a generic definition of software quality in terms of six quality factors, which further covers some sub factors:

- **Functionality**–Suitability, accuracy, interoperability, compliance and security.
- **Maintainability**–Analyzability, changeability, stability and testability.
- **Usability**–Understandability, learnability, operability and likeability.
- **Efficiency**–Time-behaviour and Resource-behaviour.
- **Portability**–Adaptability, replaceability, installability, conformance and coexistence.
- **Reliability**–Maturity, recoverability, availability and fault-tolerance.

It defines the internal and external quality characteristics of the software product. The main disadvantage of this model is how these characteristics are to be measured.

3. Component Based System Development

A component is [11,12] a language neutral, independently implemented package of software services, delivered in an encapsulated and replaceable container, accessed via one or more published interfaces. While a component may have the ability to modify a database, it should not be expected to maintain state information. A component is neither platform-constrained nor is it application-bound.

Following the success of the structured design and OO paradigms, Component-Based Software Development (CBSD) has emerged as the next revolution in software development [13].

The idea behind CBSD is integrate the reusable components to develop the final product. This strategy reduces the development effort, time-to-market and cost. Quality and reliability come as inherited features in the product developed using CBSD approach, as the components used are time tested.

4. Quality of Component based software Development

In software product development using CBSD, the programmer's role is to integrate the pre-existing software components. The various software quality models discussed above are generic models. These models do not address the quality issues for CBSD. Bertoa [14] proposed a quality model for component based software development that allows an effective assessment of COTS components. This quality model is based on ISO 9126, adapted to deal with specific characteristics of COTS components. In the proposed model, five of the characteristics of the ISO 9126 model have been retained while the sixth characteristic, namely Portability, has been removed. Additionally, suitability and analyzability sub-characteristics have also been removed and two new sub-

Table 2. Dormey's quality model

Software product	**Implementation**			
Software product quality	**Correctness**	**Internal**	**Contextual**	**Descriptive**
Software quality attributes	Functionality, reliability	Maintainability Efficiency, reliability	Maintainability Reusability, Portability, Reliability	Maintainability Reusability, Portability, Usability

characteristics, namely compatibility and complexity, have been introduced (**Table 3**). The sub-characteristics have been further classified into two different categories, namely run time and life cycle sub-characteristics. Some of the characteristics (Usability) and sub-characteristics (learnability, understandability and operability) have changed meaning in the proposed model.

However, this model fails to validate itself on any application. Adnan & Bassem [15] has also done excellent work on developing a quality model for evaluating COTS components that support a standard set of quality characteristics. This model is developed based on study done on the six generic quality models. Staring with ISO 9126 model as a base, necessary changes have been made in it to customize it to suit COTS based development. A new characteristic, viz. Stakeholders, is proposed in this new model. It depends upon who are the members of the team responsible for developing, maintaining, integrating and/or using information system based on COTS systems (**Figure 1**).

The major drawback of this model is that it fails to explain how to measure the internal quality characteristics. Ali, Gafoor & Paul [P] proposed a new approach using a set of 13 system-level metrics, which are divided into 3 categories viz. Management, Requirement and Quality (**Table 4**).

This metrics helps managers select between appropriate components from a repository of software products and aid them in deciding between using COTS components or developing new components. This model, however, does not say much about how these metric are measured/quantified. Jasmine & Vasantha [Q] propose Defect Removal Efficiency–a quality metric that provides benefits at both project and process levels. They redefined the basic definition of defect removal efficiency in terms of the phases involved in reuse-based development and proposed a systematic approach in the defect removal process. Sharma *et al.* [R] proposed a quality model based on ISO9126 that defines the characteristics and sub-characteristics of the component and proposes to add some more sub-characteristics to it, which may be relevant in the CBSD context. This model can also be used to estimate the efforts required to achieve the required value of any characteristic.

5. Untouched Quality Related Issues in Component Selection Process

In an application developed using COTS approach, various components are integrated with each other to build the whole system of desired quality. This integration process is the most crucial part of the CBSD. There may be more than one component available in the repository or more then one vendor available for the same component that performs the desired task. Each individual component from the different sources has different quality attributes. When these individual components are integrated to develop a whole system, the quality attributes of components may change and affect the quality of the overall system.

There is an absence of any kind of metrics that can help in selecting between different components from different sources with the same functionality by evaluating relevant quality parameters of the component, such as cost-avoidance, reliability, productivity etc.

There is an increasing need for metrics meant specifically for component-based software, to help foster and manage quality in component-based software development. This is because traditional software product and process metrics are neither suitable for nor sufficient in managing the complexity of software components, which is necessary for quality and productivity improvement within

Table 3. Quality model for COTS components by Bertoa

Characteristic	Functionality	Reliability	Usability	Efficiency	Maintainability
Runtime Sub-characteristics	Accuracy, Security	Recove-rability		Time Behavior, Resource Behavior	
Life Cycle Sub-characteristics	Suitability, Interoperability, Compliance, Compatibility	Maturity	Learn ability, Understand-ability, Operability, Complexity		Changeability, Testability

Table 4. System level metric for component based system

Category	*Management*	*Requirements*	*Quality*
Metric	Cost Time to market Software engineering environment System resource utilization	Requirements Conformance Requirements stability	Adaptability Complexity of interfaces and integration Integration test coverage End-to-end test coverage Fault profiles Reliability Customer satisfaction

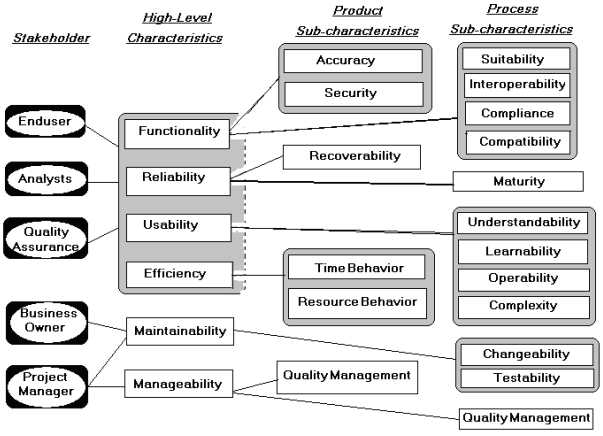

Figure 4. The new quality model for COTS-based systems

organizations adopting component-based software development.

6. Conclusions

This paper presents a survey of various generic quality models as well as quality models for COTS development. Based on this survey, some unanswered issues related to CBSD have been identified.

Future work in the development of CBSD research could include determination of quality metrics for components selection where there is more then one component available for the same task. This metric would help the developer/integrator in selecting between various components or decide upon develop the component from scratch. This metric should be easy to calculate and be feasible for practical use. The field of quality attribute determination of component-based system is extensive and more research can and should be performed in this field. This will help in increasing confidence in the use of research results, to solve problems in practical industrial settings.

REFERENCES

[1] J. A. Borretzen, "The Impact of Component-Based Development on Software Quality Attributes," http://www. toodoc.com/Software-quality-attribute-pdf.html

[2] "IEEE Standard Glossary of Software Engineering Terminologies," *IEEE Standard* 610.12-1990, 10 December 1990.

[3] J. A. McCall, P. K. Richards and G. F. Walters, "Factors in Software Quality," *Griffiths Air Force Base*, New York Rome Air Development Center Air Force Systems Command, 1977.

[4] J. A. McCall, P. K. Richards and G. F. Walters, "Factors in Software Quality," *National Technology Information Service*, Vol. 1, No. 2-3, 1977.

[5] B. W. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G. McLeod and M. Merritt, "Characteristics of Software Quality," North Holland, 1978.

[6] B. W. Boehm, W. Barry, J. R. Brown and M. Lipow, "Quantitative Evaluation of Software Quality," *International Conference on Software Engineering, Proceedings of the* 2*nd International Conference on Software Engineering*, San Francisco, 1976.

[7] R. G. Dromey, "Concerning the Chimera (Software Quality)," *IEEE Software*, Vol. 13, No. 1, 1996, pp. 33-43.

[8] R. G. Dromey, "A Model for Software Product Quality," *IEEE Transactions on Software Engineering*, Vol. 21, No. 2, 1995, pp. 146-163.

[9] ISO 9126, "Information Technology-Product Quality-Part1: Quality Model," *International Standard* ISO/IEC 9126, International Standard Organization, June 2001.

[10] ISO 9126, "Information Technology-Software Product Evaluation-Quality Characteristics and Guidelines for their Use," ISO, December 1991.

[11] M. Sparling, "Is there a Component Market?" www. cbd-hq.com/articles/2000/000606ms_cmarket.asp

[12] M. Sparling, "Lesson Learned through Six Years of Component Based Software Development," *Communications of the Association for Computing Machinery*, Vol. 43, No. 10, October 2000, pp. 47-53.

[13] P. Vitharana, "Risk & Challenges of Component Based Software Development," *Communications of the Association for Computing Machinery*, Vol. 46, No. 8, August 2003, pp. 237-252.

[14] M. Bertoa and A. Vallecillo, "Quality Attributes for COTS Components," *Proceedings of the* 6*th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering* (*QAOOSE*), Malaga, 2002.

[15] A. Rawashdeh and B. Matalkah, "A New Software Quality Model for Evaluating COTS Components," *Journal of Computer Science*, Vol. 2, No. 4, 2006, pp. 373-381.

