ISSN No. 0976-5697

Volume 2, No. 2, Mar-Apr 2011

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

# Assessing Quality Issues in Component Based Software Development

Fernando L. F. Almeida*
Department of Informatics Engineering
Faculty of Engineering of University of Porto, FEUP
Rua Dr. Roberto Frias s/n, 4200-465 Porto, Portugal
almd@fe.up.pt

Catalin M. Calistru
Innovation and Development Centre
Higher Institute of Gaya, ISPGaya
Av. dos Descobrimentos 333, 4400-103 V.N.Gaia, Portugal
cmc@ispgaya.pt

***Abstract:*** One of the most critical processes in component based software development (CBSD) is the choice of suitable commercial off-the-shelf components (COTS) that meet the user requirements. An important step in the component selection process is the evaluation of components using quality models. This paper presents the most relevant of current quality models proposed in the literature. A comparative analysis among them was performed and some issues related to CBSD were identified. Additionally, the main benefits and limitations associated with each quality model were highlighted and explored.

***Keywords:*** component based software engineering (CBSE); commercial-off-the-shelf (COTS); software quality; software integration; quality metrics

## I. INTRODUCTION

Component based software development (CBSD) model is based on the idea to develop software systems by choosing appropriate off-the-shelf components and then to assemble them with a well-defined software architecture. Because the new software development paradigm is very different from the traditional approach, quality assurance (QA) for component-based software development is a new and interesting topic in the software engineering community [1].

The objective of CBSD is to develop large systems, incorporating previously developed or existing components, thus cutting down on development time and costs. It can also be used to reduce maintenance associated with the upgrading of large systems. It is assumed that common parts (be it classes or functions) in a software application only need to be written once and re-used rather than being re-written every time a new application is developed.

CBSD embodies the "buy, but don't build" philosophy. Due to the extensive use of components, the CBSD process is quite different from the traditional waterfall approach. The waterfall approach considers that the software development method is linear and sequential [2]. Once a phase of development, the development proceeds to the next phase and there is no turning back. Instead, CBSD not only requires focus on system specification and development, but also requires additional consideration for overall system context, individual components properties and component acquisition and integration process.

The CBSD generally includes the following fundamental software development principles:

- Independent software development – large software systems are necessarily assembled from components developed by different people. To facilitate independent development, it is essential to decouple developers and users of components through an abstract and implementation-neutral interface specification;
- Reusability – while some parts of a large system will necessarily be used for specific purposes, it is essential to design and assemble pre-existing components in developing new components;
- Software quality – a component or system needs to be shown to have desired behavior, either through logical reasoning, tracing, and/or testing. Besides that, the quality assurance approach must be modular to be scalable;
- Maintainability – a software system shall be understandable and easy to evolve.

This work presents a survey of the most relevant quality models for commercial off-the-shelf (COTS) components. Further, this work extracts the major benefits and limitations of each model and a critical comparison among them is made. Section II introduces the CBSE methodology. Section III presents the major benefits and difficulties of CBSE approach. Section IV analysis the requirements of a quality model for software-based components. Section V presents the most relevant quality models for COTS components founded in literature. Finally, section VI compares the various quality models presented in section before and section VII draws conclusions.

## II. THE CBSE CONCEPT

The Component based software engineering (CBSE) is a branch of software engineering which is concerned with development of software systems based on existing in-house and/or COTS components. Reusing previously developed components in developing software has many benefits, mainly in terms of reduced costs and time to market [3]. Further, since a component is repeatedly used, it undergoes repeated testing when used in different systems and operating environments. This will increase the quality of a stand-alone component as well as the systems where it is being used. This can greatly help in materializing the benefits of standard domains. A standard domain can always provide well-defined standard components, which can be easily reused in any case. Therefore, component technology heavily supports code reuse, which was previously not the case.

CBSE approaches software development in a way, which is very different from other previous approaches such as procedural and object-oriented approaches. In these approaches, often, the same team is responsible for developing the classes and procedures that develops a software system. This approach is not very efficient in developing high quality large and complex systems in minimum time [3]. Further, procedural or object-oriented approaches for these systems often may lead to slippage of time schedule, and consequently, failure of projects. CBSE is supposed to be an effective approach for big and complex systems. Peculiarity with CBSE is that it approaches software development in a way which is similar to that adopted by other popular engineering streams for manufacturing products [4]. In fact, all of them develop products from highly reusable standardized smaller units or parts.

## III. BENEFITS AND DIFFICULTIES

### A. *Major Benefits*

The obvious benefit in the use of CBSE approach is the time-to-market, thus lowering the cost of developing the software. Shorter development cycles will save time as to developing a system from scratch. Developing software systems using CBSE offer many advantages, namely:

- Developing costs are reduced since existing components are used to develop the system;
- Reliability is increased since the components have previously been tested in various contexts;
- Time to market is reduced since the components used already exist;
- Maintenance costs are also reduced since the followed development process is well standardized;
- Efficiency and flexibility is improved due to the fact that components can easier be added or replaced.

The figure 1 illustrates the key goals and advantages of CBSE approach.

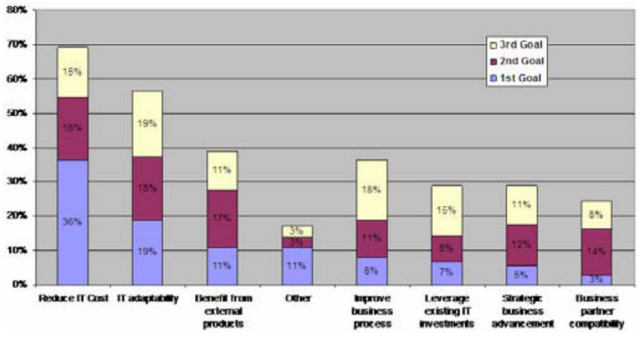

Figure 1. Key goals and advantages in CBSE

The main tangible benefits are shorter development life cycle and reduction in IT costs. There are also some intangible benefits like IT adaptability, improved business processes and benefits from external products.

### B. *Major Difficulties*

The CBSE approach typically requires the establishment of an initial component-based architecture, where component interactions are determined for each system interface operation and component object architecture constraints and interfaces are defined as needed.

In the following is a brief overview of a non-exhaustive list of problems that may be faced by component users:

- Ensuring version compatibility – versions of components may change during the course of evolution of the target application. Therefore, the management of version numbers for the applications is more complex because of the availability of multiple component versions;
- Understanding components – program understanding is important for software engineering activities such as software reuse and maintenance. Components have important aspects that need to be understood. UML models showing package and deployment diagrams can be used for this task;
- Selecting components – based on the functional, non-functional and business requirements of an application, developers may need to select a component from a set of available components. The component needs not only to meet the functional requirements, but also needs to possess the appropriate mix of quality attributes, and have no adverse impact on the quality of the target application;
- Verifying and validating component assemblies – as with any software development process, verification and validation are important activities in CBSD. Since components are mostly black-box, validation of the component assembly cannot make use of code-based test generation techniques. They need to rely on the specifications, which may be available in various degrees of detail, or as part of the API (Application Programming Interface);
- Performing regression testing on component software – regression testing is a costly process and often testers use minimization techniques to reduce the amount of re-testing. However, a lack of relevant testing information in components makes it difficult to effectively determine the test cases that should be rerun. A metadata approach with the description of components can be used to address the problem of test case selection [5];
- Maintaining applications using software components – software maintenance may be necessitated because of changes in components (versions, availability of superior components, etc), or changes in the requirements of the application and a host of other reasons. The process of maintenance may require the development of new wrappers for existing components, or the procurement of new components.

## IV. THE QUALITY MODEL FOR SOFTWARE COMPONENTS

### A. *The notion of quality in software development*

According to the IEEE Standard Glossary of Software Engineering Terminology, software quality is defined as “the degree to which a system, system component, or process meets specified requirements” and also as “the degree to which a system, system component, or process meets customer or used needs or expectations”. In fact, software quality can either refer to the software product quality or the software process quality. Examples used for the product and process quality are the CMM-I (Capability Maturity Model-Integrated) and ISO standards.

According to Ljerka Dukic and Jorgen Boegh [6] software quality model is defined as a set of characteristics and relationships between them that provide the basis for specifying quality requirements and evaluating quality. As a consequence, a software quality model is a good tool to evaluate the quality of a software product [7].

A more concise definition of a quality model is presented by Donald FireSmith [8], which defines a quality model as a hierarchical model (i.e. a layered collection of related abstractions and simplifications) for formalizing the quality of a system in terms of its factors, sub-factors, criteria and measures as described below:

- Quality characteristics – high level characteristics of a system that capture major aspects of its quality (e.g., functionality, performance or reliability);
- Quality sub-characteristics – major components of the quality factors mentioned above that capture a subordinate aspect of the quality of the systems (e.g., accuracy or expansibility);
- Quality attributes – specific descriptions of a system that provide evidence for or against the existence of a specific quality factor or sub-factor;
- Quality measures – make quality attributes measurable, objective and unambiguous.

Software Product quality assessment can either be white box or black box assessment. White box assessment is based only on the source code, but source code will not be available for certain categories of software components like COTS components and reusable COTS components. On the other side, black box quality assessment is based on reliability testing and the evaluation of externally viable characteristics that can be done using quality models that can also be applied to all kinds of software components. Since, most software components are black boxes, quality models are therefore needed to evaluate the quality of software components.

### B. *Influence of quality in software components*

Several factors influence the quality of component-based applications. Benchman et al. state that software quality depends on the quality of its components and on the component framework used [9]. Woodman et al. add that the development process and the maturity of an organization also influence the quality of component-based software products. In software systems built by assembling components, it is easy to perceive that the quality of its components, directly or indirectly, influences the quality of the final software.

The quality of the final achieved software is highly influenced by the 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 [10]. A large range of components, which perform the same function, are available from different vendors. This makes it very difficult 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 highly 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.

Several authors such as Sergey Berezin and Edmund Clarke consider that the properties of a system are influenced more by the interaction of its components than by the properties of a single component [11]. Crnkovic et al. proposed the prediction theory. This theory consists of predicting the properties of the assemblies of components before they are acquired and used. This theory is based on assumptions about the environment in which the assembly will run and on information about the components involved. The research model for the prediction theory suggests that component properties, directly or indirectly, involved in any research need to be defined along with the means that may be established and measured [12].

## V. PROPOSED QUALITY MODELS

All the existing software quality models already proposed are based on the ISO 9126 specification and are extensions or modifications of the model customized to suit the software component domain.

ISO 9126 provides the definition of the characteristics and associated quality evaluation process to be used when specifying the requirements for and evaluating the quality of software products throughout their life cycle. It is important to refer that this standard does not provide subcharacteristics and metrics, nor the method for measurement, rating and assessment. ISO 9126 sets out the following six quality characteristics, which are intended to be exhaustive [13]:

- Functionality is the set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs;
- Reliability is the set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time;
- Usability is the set of attributed that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users;
- Efficiency is the set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used under stated conditions;
- Maintainability is the set of attributes that bear on the effort needed to make specified modifications;
- Portability is the set of attributes that bear on the ability of software to be transferred from one environment.

An analysis and description of different proposed quality models will be made in the next sections.

### A. *FURPS Quality Model*

The FURPS model originally presented by Robert Grady in 1992 is structured in basically the same manner as the ISO 9126 specification [14]. Then, it had been extended by IBM Relational Software into FURPS+, where the "+" indicates such requirements as design constraints, implementation requirements, interface requirements and physical requirements.

The FURPS supports the following characteristics depicted in figure 2.

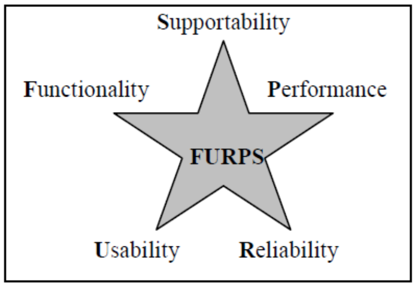

Figure 2. Characteristics of FURPS quality model

FURPS stands for:

- Functionality – which may include feature sets, capabilities and security;
- Usability – which may include human factors, aesthetics, consistency in the user interface, online and context sensitive help, wizards and agents, user documentation, and training materials;
- Reliability – which may include frequency and severity of failure, recoverability, predictability, accuracy, and mean time between failure (MTBF);
- Performance – imposes conditions on functional requirements such as speed, efficiency, availability, accuracy, throughput, response time, recovery time, and resource usage;
- Supportability – which may include testability, extensibility, adaptability, compatibility and instability.

The FURPS categories are two different types: Functional (F) and Non-functional (URPS). These categories can be used as both product requirements as well as in the assessment of product quality.

## B. *Dromey Quality Model*

Dromey proposed a working framework for building and using a practical quality model to evaluate requirement determination, design and implementation phases [16]. Dromey points out that high level quality attributes, such as maintainability, functionality and reliability, cannot be built into the system. The alternative way to input quality into software is by identifying a set of properties and build them up consistently, harmoniously and fully to provide high level quality. Additionally, links must be established between tangible product properties and intangible quality attributes.

Five steps for quality model were constructed and refined. Dromey includes high-level quality attributes such as: functionality, reliability, usability, portability, reusability and process-mature. In comparing to ISO 9126, additional characteristics like process maturity and reusability are noticeable. Subattributes associated with reusability are machine-independent, separable and configurable; while process maturity includes client-oriented, well-defined, assured and effective attributes. Process maturity is an attribute which has not been considered in the previous models.

## C. *Bertoa's Quality Model*

This model was proposed by F. Bertoa and Antonio Valecillo in 2002. The motivation behind this model was to make an attempt to define the attribute that can be described by COTS vendors (no matter whether they are external or internal providers) as part of the information provided by them. These attributes were supposed to be an aid in the COTS components quality assessment and selection procedure carried out by software designers and developers.

The ISO 9126 quality model was refined and customized to accommodate particular characteristics of COTS components. Different kinds of quality characteristics that could be applied to COTS components were identified. The characteristics were also discriminated into local characteristics (for individual components), global characteristics (at the software architecture level), runtime characteristics and product characteristics.

An overview of the Bertoa's model is depicted in table I.

Table I. Vision of Bertoa's model [17]

| Characteristics | Sub-characteristics |
|---|---|
| Functionality | Accuracy, security, suitability, interoperability, compliance and compatibility |
| Reliability | Recoverability and maturity |
| Usability | Learnability, understandability, operability and complexity |
| Efficiency | Time behavior and resource behavior |
| Maintainability | Changeability and testability |

The metrics used in this model are:

- Presence – identifies whether an attribute is present in a component;
- Time – this metric is used to measure time intervals;
- Level – this metric is used to indicate the degree of effort or ability;
- Ratio – this metric is used to measure percentages.

This model encompasses all the characteristics already presented in the ISO 9126 model. This means that not many component specific changes were made to the original model except the addition of attributes like complexity and compatibility.

## D. *Ali, Gafoor & Paul Quality Model*

Ali, Gafoor & Paul proposed in 2003 a quality model for evaluating COTS components that support a standard set of quality characteristics. It consists in a new approach using a set of 13 system-level metrics, which are divided into three categories: Management, Requirement and Quality.

An overview of the Ali's metrics is given in table II.

Table II. Vision of Ali's metrics [18]

| Category | Metric |
|---|---|
| Management | Cost, time-to-market, software engineering environment and system resource utilization |
| Requirements | Requirements conformance and requirements stability |
| Quality | Adaptability, complexity of interfaces and integration, integration test coverage, end-to-end test coverage, fault profiles, reliability and customer satisfaction. |

There are three metrics that deserve a special attention due to their level of novelty. The Software Engineering Environment is a metric that intends to measure the capability of producing high quality software. The End-to-End (E2E) Test Coverage complements the integration test coverage metric and it is responsible to return the fraction of the system that has undergone satisfactory E2E testing. Finally, Customer Satisfaction gives the degree to which the software has met customer expectations.

These 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.

## E. *Alvaro's Quality Model*

The next component quality model was proposed by A. Alvaro, E. Santana and S. Meira in 2005 [17]. The purpose of this model is to determine which quality characteristics should be considered for the evaluation of software components. The proposed quality model is also part of a Software Component Certification Framework that was being investigated with the objective of acquiring quality in software components that will be stored in repository systems.

The model follows the ISO 9126 standard like the previous models. However, some changes have been made in order to develop a consistent model to evaluate software components. A few new sub characteristics have been added and existing sub characteristics have been removed.

The added sub characteristics include self contained, configurability, scalability and reusability. Self contained is an intrinsic property of a component and must be analyzed. Configurability becomes essential for the developer to analyze if the component can be easily configured. Scalability is relevant in order to express the ability of the component to support major data volumes. Reusability is important for the reason that software factories have adopted component based approaches on the premise of reuse.

On the other side, the maintainability characteristic and analyzability sub characteristic has been removed from the ISO 9126 model. Another change is that the installability sub characteristic has been changed to deployability.

The Alvaro's model has also added another high level characteristic called business with the following sub characteristics: development time, cost, time to market, targeted market and affordability.

### *F. Adnan Rawesdah's Quality Model*

Adnan Raweshdah and Bassem Matalkah build in 2006 a new model that supports standard set of quality characteristics suitable for evaluating COTS components along with newly defined sets of sub characteristics associated with them [19]. The model also attempts to match the appropriate type of stakeholder with the corresponding quality characteristic, which is a feature that is missing in the already existing models described above.

An outline of Adnan Rawesdah's model is given in table III.

Table III. Vision of Adnan Rawesdah's model [17]

| Characteristics | Sub-characteristics |
|---|---|
| Functionality | Accuracy, security, suitability, interoperability, compliance, compatibility and self-contained |
| Reliability | Recoverability and maturity |
| Usability | Learnability, understandability, and operability |
| Efficiency | Time behavior, resource behavior and scalability |
| Maintainability | Changeability and testability |
| Manageability | Quality management |

The sub characteristics fault tolerance, configurability, scalability and reusability have been removed. However, a new characteristic manageability has been added. The meaning of manageability is concerned with developing and refining estimates of effort and deadlines for the project as a whole. A new sub characteristic Quality Management which indicated the people within the organization, who are constantly finding out ways to improve quality of operation, product, budgets, schedule and services offered by the firm is also added.

The model identifies the following stakeholders:

- End user – interacts with the system;
- Analyst – produces the business model;
- QA Officer – tests and validated the product;
- Project Manager – constructs and manages the process.

### *G. Sharma Quality Model*

Sharma et al. proposed in 2007 a quality model based on ISO 9126 that defines the characteristics and sub-characteristics of the components and proposes to add some more sub-characteristics to it, which may be relevant in the CBSD context. The most innovative part of this model is that it can also be used to estimate the effort required to achieve the required value of any characteristic.

A particular attention is given for the evaluation of reusability metric. The reusability can be measured directly using the Reuse Leverage for Productivity (RL) metric, as it follows [20]:

$$RL = \frac{\text{Productivity with Reuse}}{\text{Productivity without Reuse}} * 100 \qquad (1)$$

This notion of reuse leverage can only be measured after the introduction of reuse. It is desirable to be able to predict the effect of reuse, which can be obtained either to work initially on some pilot projects or to work with estimated or with industry benchmarks.

Reusability can also be measured indirectly. Complexity, adaptability and observability can be considered as a good measure of reusability indirectly. The work conducted by Sharma et al. considers two approaches to measure the reusability of a component. The first is a metric that measures how a component has reusability and may be used at design phase in a component development process. This metric, Component Reusability (CR) is calculated by dividing the sum of interface methods providing commonality functions in a domain to the sum of total interface methods. The second approach is a metric called Component Reusability Level (CRL) to measure particular component's reuse level per application in a component based software development. This metric is again divided into two sub-metrics. First is the CRLLOC, which is measured by using lines of code, and is expressed as percentage as given as [20]:

$$CRLLOC\ (C) = \frac{Reuse\ (c)}{Size\ (c)} * 100 \qquad (2)$$

In the equation (2), Reuse(c) is the lines of code reused component in an application and Size(c) is the total lines of code delivered in the application.

The second sub-metric is CRLFunc, which is measured by dividing functionality that a component supports into required functionality in an application. This metric gives an indication of higher reusability if a large number of functions used in a component.

### *H. Jasmine & Vasantha Quality Model*

Jasmine & Vasantha proposed in 2008 a Defect Removal Efficency quality metric that provides benefits at both projects and process levels [10]. 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.

The model introduced a metric to reduce defect injection and boost defect removal efficiency. Defect removal efficiency (DRE) metric quantifies the excellence of the product by computing the number of defects before release of the product to the total number of latent defects. DRE is defined as the number of defects removed during development phase divided by the total number of latent defects. DRE depends upon time and method used to remove defects. Anyway, it is always more lucrative for defects to be prevented rather than detected and eliminated.

Certain amount of defects can be prevented through error removal techniques like educating development team through training, by use of formal specifications and formal verifications. It can also be prevented with use of tools, technologies, process and standards. Several tools are available right from requirements phase to maintenance phase to automate the entire development process. By inculcating quality standards in software development, defects can be prevented to a maximum extent.

## VI. EVALUATION OF QUALITY MODELS

There are a considerable number of quality models in software engineering literature, each one of these quality models consists of a number of quality characteristics (or factors, as called in some models). These quality characteristics could be used to reflect the quality of the software product from the view of that characteristic. Selecting which one of the quality models to use is a real challenge. In order to help this choice the quality models presented in section above are compared in table IV.

Table IV. Main benefits and limitations

| Model | Benefits | Limitations |
|---|---|---|
| FURPS | Easy to use<br>Well structured | Portability is not considered |
| Dromey's | Considers software quality attributes<br>Includes the portability dimension | Efficiency is not considered |
| Bertoa's | Addition of attributes such as complexity and compatibility | Attributes such as interfaces, versions and reusability are not considered |
| Ali's | Introduction of 13 system-level metrics | Lack of information about how to measure and quantify these metrics |
| Alvaro's | Addition of attributes such as self contained, configurability and scalability<br>Redefinition of usability concept | Reusability and testability attributes are not sufficiently described |
| Adnan's | Addition of manageability attribute | Lack of innovative concepts<br>Some important sub characteristics for COTS components were eliminated |
| Sharma's | Introduction of new metrics for the evaluation of reusability | It is not a full quality model |
| Jasmine's | Addition of a new quality metrics called defect removal efficiency | It is not a full quality model |

FURPS is the oldest quality model that can be used to assess the COTS quality components. It is easy to use and well structured in two categories of requirements: functional requirements and non-functional requirements. The main disadvantage of the FURPS model is it does not take into account the portability of the software product, which is an important requirement of the COTS components.

Dromey's model presented the innovation to connect software product quality with software quality attributes. The model introduced software product quality parameters such as correctness, internal, contextual and descriptive. Besides that, the portability as a software quality attribute was added comparing to the FURPS model. The disadvantage of this model is that efficiency of software is not considered for determining the quality of software.

Bertoa's model encompasses all the characteristics already present in the Dromey's and ISO 9126 models. This means that not many component specific changes were made to the original model except the addition of attributes like complexity and compatibility. The mains disadvantages of this model are that it does not discuss about significant component characteristics like interfaces, versions and reusability. However this was an important step taken in the direction of designing quality model especially suitable for components and the later models proposed are all based on Bertoa's model.

Ali, Gafoor & Paul model proposed a new approach using a set of 13 system-level metrics such as system resource utilization, requirements stability and end-to-end test coverage organized in three categories (management, requirements and quality). These metrics helps managers choose 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 metrics are measured and quantified.

Alvaro's model can be considered a huge step forward, even though it is similar to Bertoa's model, but it added a number of components specific quality characteristics like self contained, configurability, and scalability. Another advantage in this model is that it has redefined usability in the component context, which means its ability to be used by the application developer when constructing a software product or system with it. However, this model has a few drawbacks. The first is that reusability has been recognized only as a quality attribute and not as a quality factor. According to Hopkins Jon reusability, by itself, should be placed as quality factor together with its own set of quality sub characteristics, attributes and measures [21]. Additionally, not enough importance is given to testability, which again is included only as a quality sub characteristic. Typically, when a COTS component is purchased from a vendor, it is basically a black box. The buyer usually receives an executable version of the component and a description of its functionality. The testers have to make use of this little information to test if the component is compatible and working. Hence, a higher testability degree increases the chance of discovering errors, which will increase the quality of the component. Therefore, the inclusion of testability as a high level quality characteristic would be useful.

Adnan Raweshdah's model doesn't bring substantial improvements. The sub characteristics fault tolerance, configurability, scalability and reusability were eliminated from Alvaro's model in order to simplify the model and increase the industrial adoption of COTS components quality models. However, a new main characteristic level called Manageability, which refers to quality management has been introduced. Some critiques can be formulated to this model particularly in terms of innovation, relevance and quality measures. First, the eliminated sub characteristics are very significant to COTS components. Besides that, the new Manageability characteristic is more process specific rather than product specific and quality attributes. Additionally, the authors have also missed to describe the quality attributes and measures of their quality model.

The Sharma and Jasmine models can't be seen as complete quality models. They just add some metrics to help managers selecting the best appropriate COTS components. Sharma model made a significant effort for the evaluation of reusability, which is a fundamental characteristic of a COST component. On the other side, Jasmine model proposed a Defect Removal Efficiency measure, which is a quality metric that provides benefits at both project and process levels.

## VII. CONCLUSIONS

The growing use of commercial products in large systems makes evaluation and selection of appropriate products an increasingly essential activity. However, many organizations struggle in their attempts to select an adequate product for use in Component-Based Software Development (CBSD), which is being used in a wide variety of application areas and the correct operations of the components are often critical for business success. In this way, assessment and evaluation of software components has become a compulsory and crucial part of any CBSD lifecycle. The software components quality evaluation has become an essential activity in order to bring reliability in (re)using software components.

In this sense, a survey of various quality models for COTS development was conducted. All the proposed models follow the ISO 9126 quality model framework with a few corrections and adjustments made to suit the software component domain.

A comparative analysis among them was also performed and some issues related to CBSD were identified. Currently the most widely cited and used quality models are the Bertoa's and Alvaro's model, which are enough complete to evaluate the major quality aspects of CBSD methodology. Additionally, more recent quality models proposed by Sharma and Jasmine have a big potentially to be adopted. They bring new metrics in terms of reusability and defect removal efficiency that provides important benefits at both project and process levels.

Future work in the quality models of CBSD research could include the impact of lack information in the measurement of a component quality. It is common not to be possible to measure some of the attributes due to a number of reasons, essentially because not all the information is provided by the component vendors. Besides that, it would be interesting to suggest a comprehensive model specific for COTS components with a minimum set of attributes. Additionally, with regard to the quality attributes of components, it is very important to interact and work with independent parties and organizations that help evaluating them, in order to obtain trustworthy measures. This would greatly simplify, or eliminate, most of the tests that need to be performed again by the components acquirer, and that heavily reduce the potential benefits of using components.

## VIII. REFERENCES

[1] I. Kaur, P. Sandhu, H. Singh and V. Saini, "Analytical Study of Component Based Software Engineering", World Academy of Science, Engineering and Technology, vol. 50, 2009, pp. 437-442.

[2] X. Zhang, T. Hu, H. Dai and X. Li, "Software development methdlogies, trends and implications", Proc. of the Southern Association for Information Systems Conference, 2010, pp. 173-178, doi:10.3923/itj.2010.1747.1753.

[3] S. Pandeya and A. Tripathi, "Testing component-based dostware: what it has to do with design and component selection", Journal of Software Engineering and Applications, vol. 1, 2011, pp. 37-47.

[4] A. Dogru and M. Tanik, "A process model for component-oriented software engineering", IEEE Software, vol. 20, issue 2, 2003, pp.34 -41, doi:10.1109/MS.2003.1184164.

[5] M. Harrold, A. Orso, D. Rosenblum, G. Rothemel, M. Soffa and H. Do, "Using component metadata to support regression testing of component-based software", Proc. of the International Conference on Software Maintencance, 2001, pp. 716-725.

[6] L. Dukic and J. Boegh, "COTS Software Quality Evaluation", Proc. of International Conference on COTS Based Software Systems, 2003, pp.72-80, doi:10.1007/3-540-36465-X_7.

[7] K. Sacha, "Evaluation of Software Quality", Proc. of Conference on Software Engineering: Evolution and Emerging Technologies", 2005, pp. 381-388.

[8] D. FireSmith, "Archieving quality requiremnts with reused software components: challenges to successful reuse", Proc. of Second International Workshop on Models and Processes for the Evaluation of off-the-shelf Components (MPEC'05), 2005, pp. 16-29.

[9] O. Preiss and A. Wegmann, "A systems perspective on the quality description of software components", Proc. of the 6th World Multiconference on Systemics Cybernetics and Informatics, vol. 7, 2002, pp. 250-255.

[10] J. Pande, R. Bisht, D. Pant and V. Pathak, "On some quality issues of component selection in CBSD", Journal of Software Engineering & Applications, vol. 3, 2010, pp. 556-560, doi:10.4236/jsea.2010.36064.

[11] S. Berezin, S. Campos and E. Clarke, "Compositional Reasoning in Model Checking", Lecture Notes in Computer Science, vol. 1536, 1998, pp. 81-102.

[12] I. Crnkovic, H. Schmidt, J. Stafford and K. Wallnau, "Anatomy of a research project in predictable assembly", Proc. of the 5th ICSE Workshop on Component-based Software Engineering, 2002, pp. 74-79.

[13] F. Losavio, L Chirinos, N. Lévy and A. Ramdane-Cherif, "Quality Characteristics for Software Architecture", Journal of Object Technology, vol. 2, no. 2, 2003, pp. 133-150.

[14] P. Berander, L. Damm, J. Eriksson, T. Gorschek and D. Milicic. Software Quality Attributes and Trade-offs, 1rd ed., Blekinge Institute of Technology, Ronneby: Sweden, 2005, pp. 3-19.

[15] R. Al-Qutaish, "Quality models in software engineeting literature: an analytical and comparative study", Journal of American Science, vol. 6, no. 3, 2010, pp. 166-175.

[16] J. Yahaya, A. Deraman and A. Hamdan, "Software Quality from Behavioural and Human Perspectives", International Journal of Computer Science and Network Security, vol. 8, no. 8, 2008, pp. 53-63.

[17] S. Kalaimagal and R. Srinivasan, "A Retrospective on Software Component Quality Models", ACM SIGSOFT Software Engineering Notes, vol. 33, no. 6, 2008, pp. 2-8.

[18] S. Ali, A. Ghafoor and R. Paul, "Metrics-Guided Quality Management for Component-Based Software Systems", Proc. of the 25th Annual International Computer Software and Applications Conference (COMPSAC'01), 2001, pp. 303-308, doi:10.1.1.11.8759.

[19] A. Raweshdah and B. Matalhak, "A New Software Quality Model for Evaluating COTS Components", Journal of Computer Science, vol. 2, no. 4, 2006, pp. 373-381.

[20] A. Sharma, R. Kumar and P. Grover, "Managing Component-Based Systems with Reusable Components", International Journal of Computer Science and Security, vol. 1, issue 2, 2007, pp. 52-57.

[21] J. Hopkins, "Component Primer", Communications of the ACM, vol. 43, no. 10, 2000, pp. 27-30, doi:10.1145/352183.352198.