

## Estimation of Quality for Software Components – an Empirical Approach

**Arun Sharma**

Amity Institute of Information Technology,  
Amity University, Noida [arunsharma@aiit.amity.edu](mailto:arunsharma@aiit.amity.edu)

**Rajesh Kumar**

School of Mathematics & Computer Applications  
Thapar University, Patiala. [rainagdev@yahoo.co.in](mailto:rainagdev@yahoo.co.in)

**P. S. Grover**

Guru Tegh Bahadur Institute of Technology,  
GGS IP University, Delhi [groverps@rediffmail.com](mailto:groverps@rediffmail.com)

### Abstract

Component-Based Development (CBD) approach now is widely accepted in software industry. This approach enables efficient application development through the integration of already developed software components. The success of these applications heavily depends upon the selection of appropriate components to fit customer requirements. Therefore it is very necessary to evaluate the quality of components before using them in the final system. Quality models proposed so far can not be fully implemented as-it-is on components and component-based systems (CBS) due to architectural differences in the development approach. Present paper surveys a number of quality models for traditional and component-based systems and proposes a new model for CBS by proposing some new characteristics, which may be very relevant in the context of components. All the quality characteristics may not be of prime importance for an application to be developed for a specific domain. Therefore, it is necessary to identify only those characteristics/sub-characteristics, which may have higher priorities over the others. The present work uses Analytical Hierarchy Process (AHP) to assign the weight values to the characteristics for the proposed model. These weight values are then used to evaluate the quality contribution of sub-characteristics, characteristics and then finally the overall quality of the component by using the appropriate metrics. This approach can be used to identify and select better quality component among several others which can be used in the final system.

**Keywords:** Components, Quality, Quality Model, ISO9126, AHP

### 1.0 Introduction

CBSD has become an important alternative for building complex and distributed applications. Low cost and efforts, faster delivery, and quality are the main benefits

of this approach. However, although the market is growing rapidly in terms of the usage of this approach, very less could be done in terms of the quality aspects of this emerging approach. Quality, not only describes and measures the functional aspects of the software (what a system does), but also extra functional properties (how the system is built and performs). In CBD quality becomes more important as developers have to rely on the vendors, from whom they are taking the component/s to integrate it in the application. This component may not meet the quality requirements set by the developers and may produce catastrophic results in a typically complex situation. Therefore, quality of the components must be considered with top priority to increase the reliability of the end product.

Several quality models are proposed by researchers for software systems which include McCall, Boehm, FURPS, Dromey, Sehra, ISO 9126 and others. Most of these models are generic models and are proposed for general application systems. Out of these models, ISO 9126 is a prominent model which includes the findings of almost all other models. This is widely recognized in industry and research community. Researchers made several efforts to implement this model for component based systems with minor modifications. Present work also considers this model as base model and proposes a new model by adding/removing some of the characteristics to/from this model.

Section 2 briefs about the general concepts of quality and quality models. Section 3 defines the quality characteristics and sub-characteristics in the component context. The proposed model for component based systems is presented in section 4. Section 5 describes the methodology to evaluate this model with the help of an experiment. Paper concludes with the discussion and conclusion in Section 6 and 7 respectively.

## 2.0 Software Quality

The discipline of software quality is a planned and systematic set of activities to ensure that quality is built into the software. It consists of software quality assurance, software quality control, and software quality engineering. According to the IEEE 610.12 [1] standard, software quality is a set of attributes of a software system and is defined as:

- (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.
- (3) Quality comprises all characteristics and significant features of a product or an activity which relate to the satisfaction of given requirements.

Quality, in general, is the totality of features and characteristics of a product or a service that bears on its ability to satisfy the given needs. For example, conformance to specifications. It is the degree to which a customer or user perceives that software meets his or her composite expectations. Or in other words, it is the composite characteristics of software that determine the degree to which the software in use, will meet the expectations of the customer [2].

The evaluation of quality for a software system depends upon the following:

- 1) Quality Model
- 2) Quality characteristics
- 3) Metrics to assess the attributes of characteristics and sub-characteristics.

Quality model describes the set of characteristics and relationships between them, which provide the basis for specifying quality requirements and evaluating quality [3]. A quality model establishes a framework to perform some kind of measurement of the specific desirable features that are needed in the final system and perceived by the end user. Here an important assumption is made that internal product characteristics, related to the product development, affect external attributes or quality in use [4]. Following section briefs some of the widely used and accessed quality models for software applications.

### 2.1 McCall Model

The first quality model was proposed by McCall [5] (1977). He presented a Software Quality Factor Framework and classified the quality attributes into three groups shown in Table 1.

- Product Operation: refers to the system's ability to be quickly understood, efficiently operated and capable of providing the results required by the user – i.e. involving attributes, such as, correctness, reliability, efficiency, integrity and usability;

- Product Revision: relates to error correction and system adaptation. This aspect is generally considered the costliest part of the software and involves attributes, such as, maintainability, flexibility and testability;
- Product Transition: refers to distributed processing, together with rapidly changing hardware and involves attributes, such as, portability, reusability and interoperability.

Product operation factors	Product revision factors	Product transition factors
<ul style="list-style-type: none"> <li>• Correctness</li> <li>• Reliability</li> <li>• Efficiency</li> <li>• Integrity</li> <li>• Usability</li> </ul>	<ul style="list-style-type: none"> <li>• Maintainability</li> <li>• Flexibility</li> <li>• Testability</li> </ul>	<ul style="list-style-type: none"> <li>• Portability</li> <li>• Reusability</li> <li>• Interoperability</li> </ul>

Table 1: McCall Quality Model

The major advantage of this model is the relationship created between its quality characteristics; however the main drawback is that it does not include the functionality aspect of the software product. Also, some of the factors and measurable properties, like traceability and self-documentation among others, are not really definable or even meaningful at an early stage for non-technical stakeholders. It is, therefore, difficult to use this framework to set precise and specific quality requirements. This model is not applicable with respect to the criteria outlined in the IEEE Standard for a Software Quality Metrics Methodology for a top-down approach to quality engineering. It is, therefore, not suited as a foundation for Software Quality Engineering according to the stated premises [6].

### 2.2 Boehm's Model

The Boehm model is similar to the McCall model in that it represents a hierarchical structure of characteristics, each of which contributes to total quality. Boehm's notion includes users' needs, as McCall's does; however, it also adds the hardware yield characteristics not encountered in the McCall model [7]. Boehm's model looks at utility from various dimensions, considering the types of user expected to work with the system once it is delivered. General utility is broken down into Portability, Utility and Maintainability. Utility is further broken down into Reliability, Efficiency and Human Engineering. Maintainability is in turn broken down into Testability, Understandability and Modifiability. However, Boehm's model does not elaborate the methodology to measure these characteristics.

### 2.3 ISO 9126

ISO (International Standard Organization) proposed a standard, known as ISO 9126 [3], which provide a generic definition of software quality in terms of six main characteristics for software evaluation. These characteristics include: Functionality, Reliability,

Usability, Efficiency, Maintainability and Portability. These characteristics cover some sub-characteristics, as shown in Table 2.

Software Product Quality					
Functionality	Reliability	Usability	Efficiency	Maintainability	Portability
Subtlety	Maturity	Understandability	Time-	Stability	Adaptability
Accuracy	Fault Tolerance	Learnability	Behavior	Analyzability	Installability
Interoperability	Recoverability	Operability	Resource-	Changeability	Replaceability
Security		Attractiveness	Utilization	Testability	Co-existence

**Table 2: ISO 9126 Quality Characteristics**

One of the advantages of this model is that, it identifies the internal and external quality characteristics of a software product. On the other hand, it does not show very clearly how these aspects can be measured [8].

## 2.4 Dromey's Model

Dromey [9] proposed a quality evaluation framework that analyzes the quality of software components through the measurement of tangible quality properties. All these components possess intrinsic properties that can be classified into four categories:

- **Correctness:** Evaluates if some basic principles are violated.
- **Internal:** Measure how well a component has been deployed according to its intended use.
- **Contextual:** Deals with the external influences by and on the use of a component.
- **Descriptive:** Measure the descriptiveness of a component (for example, does it have a meaningful name?).

These properties are used to evaluate the quality of the components. While Dromey's work is interesting from a technically inclined stakeholder's perspective, it is difficult to see how it could be used at the beginning of the lifecycle to determine user quality needs. The disadvantage of the Dromey's model is also associated with reliability and maintainability. It is not feasible to judge both attributes of a system before it is actually operational in the production area [10].

## 3.0 Quality for Component Based Systems

In component-based systems, it is very difficult to relate system properties to component properties [11]. For component-based systems crucial questions in relation to choosing a quality component, are the following:

- How likely is it that the component will work properly in target application without or with minor reconfiguration?
- In what ways has the component been tested in a sufficient variety of situations and are the testing methods relevant to intended use?
- Has the component already been used in other applications similar to this, if so, how much and with what result?

- What are the implications of using this component on system performance, reliability, maintainability, portability and other aspects?

To answer these questions, we need to choose those quality attributes and finally a quality model which may be applied on software components and component based systems. There is no general consensus on the traditional quality models which can fit for component based systems. McCall's ignored functionality, Boehm's contains a diagram without any suggestion about measuring the quality characteristics, ISO-9126 does not show very clearly how the attributes can be measured. Thus, there is an absence of any kind of metrics that could help in evaluating quality characteristics objectively, in particular when the underlying software project is component-based.

However, there are several models proposed exclusively for CBS also. But most of these models are based or derived from ISO9126. Bertoa [12] defines the characteristics and sub-characteristics in the changed context of component based systems. The paper divides sub-characteristics into runtime and lifecycle categories based on their nature. It adds compatibility sub-characteristic under functionality, which indicates whether former versions of the component are compatible with its current version. The paper also proposes various attributes associated with the sub-characteristics and finally defines these attributes with the classification for the measurement of such attributes like ratio, presence, integer and time [12]. Although the paper presents a good description on quality characteristics, sub-characteristics and their measurement, it fails to perform any empirical evaluation of the attributes on any application, thus leaving the proposed work as incomplete [13].

Adnan [10] also conducted a survey on various quality models available, which includes McCall, Boehm, FURPS, Dromey and ISO 9126. It performs some tailoring on ISO9126 model by adding compatibility sub-characteristic under functionality and complexity under usability. It omits stability and analyzability from maintainability and adds manageability to it. It also adds new characteristic, named Stakeholders in its proposed model who are the members of the team responsible for developing, maintaining, integrating and/or using COTS systems. The paper performs a good comparison for various quality models; however, all the models considered are traditional models and may not fit for component-based systems, as-it-is. Moreover, the proposed model does not explain how the attributes belonging to various characteristics and sub-characteristics will be measured to finally evaluate the quality of the system.

Sedigh et. al. [14] categorized the system level metrics into management, requirement and quality. Management includes the cost, time-to-market, software engineering environment and system resource utilization. Requirements include requirements conformance and requirements stability, while quality includes adaptability, complexity of interface and integration test coverage, E2E test coverage, Fault profiles, reliability and customer satisfaction. The relationship among metrics is described using the influence diagram.

The main quality characteristics (Functionality, Usability, Efficiency, Reliability, Portability and Maintainability) are available in almost all quality models proposed so far for component based systems. However, researchers differ while choosing sub-characteristics under these characteristics. Proposed work in this paper is an extension to the work mentioned above. Following section defines the terms used for various characteristics and sub-characteristics in our proposed quality model for software components:

**Functionality:** Functionality is a set of attributes that bear on the existence of a set of functions and their specified properties [ISO, 1991]. It means that the component should provide the functions and services as per the requirement when used under the specified condition. Pre-existing components with or minimum changes will allow low cost, faster delivery of end product.

**Suitability-** Suitability tries to express how well the component fits the developer's requirements. As the exact requirement can only be known to system developer, it can not be measured by component developer during its development.

**Accuracy:** It evaluates whether the component produces the accurate results with correct precision level required by the system developer.

**Interoperability-** This sub-characteristic indicates whether the format of the data handled by the target component is compliant with any international or 'de facto' standard or convention.

**Security:** It refers how the component is able to control the unauthorized access to its provided services.

**Compliance:** This characteristic indicates if a component is conforming to any international standard or certification etc.).

**Reliability:** In general, reliability is the probability that a system or component will produce failure within a given period of time. In other words, reliability expresses the ability of the component to maintain a specified level of fault tolerance (fault frequency and fault severity), when used under specified conditions. Reusability aspect of the same component with multiple applications will increase the reliability of that component as it may be

observed here that this component would have been thoroughly tested before using it in previous applications.

**Maturity-** In component context, it deals with the number of commercial versions of the component and the time interval between each version.

**Recoverability-** It measures the ability for a component to recover from an unexpected failure (e.g. through exception handling) and also to recover the lost data along with the original performance.

**Fault Tolerance:** This sub-characteristic indicates whether the component can maintain a specified level of performance in case of faults.

**Usability:** It is the ability of a component to be understood, learned, used, configured, and executed, when used under specified conditions. Obviously, here the user of the component is application/system developer rather than end user. Therefore the usability of the component should be less complex and more reusable and developer friendly so that it can be assembled properly in the final system. Sub-characteristics of Usability are defined as under [15]:

**Understandability:** It is the capability of the component to enable the user (system developer) to understand whether the component is suitable, and how it can be used for particular tasks and conditions of use.

**Learnability:** This sub-characteristic refers the ability of the component to enable the system developer to learn the application. For example, the user documentation and the help system should be complete; the help should be context sensitive and explain how to achieve common tasks, etc.

**Operability:** the capability of the software component to enable the user (system developer) to operate and control it.

**Attractiveness:** It indicates the capability of the software component to be attractive to the user. Here as stated earlier, the user is system developer, and he/she may be more interested in the programmatic interfaces, API's defining the services provided by the components so that they can be composed and integrated with the target system rather than its attractiveness or GUI interfaces, we may omit this sub-characteristic from the proposed model.

**Compliance:** This characteristic indicates if a component is conforming to any international standard or certification etc) relating to usability. Currently, no standards have been set up for this sub-characteristic in the component context; therefore it can be omitted from the quality model for the time being.

**Efficiency:** This characteristic express the ability of a component to provide appropriate performance, relative to the amount of resources used. Efficiency is affected by the component technology, mainly through resource usage by the run-time system but also by interaction mechanism. Component can be internally optimized to improve performance without affecting their specifications. Components should be tested on various platforms to check the performance [16].

**Time Behavior:** This characteristic indicates the ability to perform a specific task at the correct time, under specified conditions.

**Resource behavior:** This characteristic indicates the amount of the resources used, under specified conditions.

**Maintainability:** This characteristic describes the ability of a component to be modified. As the component developers do not have the source code of the component, they can only adapt it, reconfigure it and finally test it before including it in the final product. The sub-characteristics under maintainability are:

**Customizability-** As mentioned earlier, system developer can only adapt, reconfigure, test and finally embed it into the system, as he is not having the source code of the component. Customizability refers to the ability to modify the component through its limited available information, like interfaces and parameters. For example in JavaBeans, components can be customized through its customizable parameters (set methods).

**Testability-** This sub-characteristic refers whether the component provides some sort of tests or test suites that can be performed to the component to check its functionality inside (or in isolation of) the final system in which the component will be integrated.

**Stability-** It refers to the component ability to handle the unexpected changes during the maintenance.

**Analyzability-** We propose to remove this sub-characteristic from the quality model. A component is developed to attend certain functionalities of the application and, rarely are developed methods for its auto analyze or to identify parts to be modified (i.e. this is the main concern of *Analyzability* characteristic, according to ISO 9126). Therefore analyzability may not be required and is removed from the model. [17]

**Portability:** This characteristic is defined as the ability of a component to be transferred from one environment to another with little modification, if required. The component should be easily and quickly portable to specified new environments if and when necessary,

with minimized porting costs and schedules. In component based development, it is a very important characteristic as a component may be used and reused in various different environments. Therefore the specification of a component should be platform independent.

**Replaceability:** This sub-characteristic indicates whether the component is backward compatible with its previous versions. This means that the new component can substitute the previous ones without any major efforts.

**Adaptability:** It refers whether the component can be adapted to different specified platforms.

**Installability-** It is the capability for a component to be installed easily on different platforms.

#### 4.0 New characteristics added to quality model

**Complexity-** Because the source code of the component is not available to the application developer, the complexity of a component in CBD, is limited to interface methods, pre and post conditions, properties and interactions available to the developer. Measuring the complexity at the initial stage is helpful during analyzing, testing, and maintaining the system. This measurement could direct the process of improvement and reengineering work. [18] proposes a complexity metric for software components which is based on the interface methods and properties, available for black box components. A complexity measure could also be used as a predictor of the effort that is needed to maintain the system. In component-based systems, functionalities are not performed within one component. Components communicate and share information in order to provide system functionalities. Because the system developer adapt the component, reconfigure it and then finally integrate it in the final system without going into the internals of the component, we propose that complexity should be added under Usability characteristic.

**Trackability-** When reconfiguring a component for changed/improved functionality, maintainers must perform a full cycle of product evaluation, integration and testing. Even if the new release of a component claims to be functionally backward compatible with an older version, there will undoubtedly be differences such as resource usage, performance, or target system requirements [19]. Therefore, it is very necessary to keep track of all the parameters of the older version so that these details can be compared with the enhanced or changed parameters. Thus, maintenance activities may be extended to include provisions for keeping a proper track of various system properties during the maintenance activities, which may include tracking system performance or resource utilization, before and after any maintenance activity, like replacement of an old component with a new version with some

added/modified functionality. This may also include the possible security violations due to some maintenance activities. Tracking not only will validate any improvement efforts, but also once an information process is presumed to be stable, tracking may provide insight into maintaining statistical control. This, in effect, will ease the overall maintenance process. Therefore trackability aspect should be added under the Maintainability characteristic [20].

**Reusability**- Reusability is the intrinsic property of Component Based Systems. Reuse is the process of adapting a generalized component to various contexts of use. It improves productivity, maintainability, portability and therefore the overall quality of the end product. A reusable component must be generic, that is, it has appropriate features that enable the re-user to create specific instances of the components to satisfy application specific requirements. We propose to add it under Functionality characteristic.

**Scalability** - Scalability expresses the ability of the component to support major data volumes. It may also be measured as the number of users or transactions it can scale to without sensible decrease in response time. It is related with the performance therefore should be included under Efficiency characteristic.

**Flexibility** – It is the efforts required to modify a component under the maintenance activity.

**4.1 New Modified Quality Model**- After tailoring the quality model for CBS, the changed characteristics in the new proposed quality model for software components is as shown in Table 3. The sub-characteristics shown in bold are newly proposed additions to the ISO9126 model.

Software Product Quality					
Functionality	Reliability	Usability	Efficiency	Maintainability	Portability
Subability	Minimality	Understandability	Time-Behavior	Stability	Adaptability
Accuracy	Fault Tolerance	Learnability	Resource Utilization	Analyzability	Installability
Interoperability	Recoverability	Operability	Scalability	Changeability	Replaceability
Security		Attractiveness		Testability	Co-existence
<b>Reusability</b>		<b>Complexity</b>		<b>Trackability</b>	
				<b>Flexibility</b>	

Table 3: New proposed Quality Model for Software Components

## 5.0 Evaluation of Proposed Model

Not all quality characteristics are of equal importance to the software product. Therefore, developers must realize that to design a successful system does not mean that the system must conform to all the software quality attributes defined, but rather the most important attributes need to be identified, specified and prioritized. Moreover, the priorities of these characteristics and sub-characteristics will vary from one application domain to another. Like for throwaway applications e.g. a real time system developed to launch a missile, maintainability will be of no use. Similarly, if the application is intended

to work only for a dedicated platform, portability aspect is not required at all. However, the financial applications will require more security, efficiency, reliability, fault tolerance and availability. Similarly, for an e-Commerce system, the important characteristics would be reliability, performance, availability, security, and maintainability (maintenance on these applications occurs continuously) [21]. Therefore, we should consider only the just enough and related quality characteristics and sub-characteristics while developing a software system for a particular domain. It will eventually reduce the overall efforts, time and cost for the development. Empirically, we can assign weight values to these characteristics and sub-characteristics based on their importance in these domains. These weight values may vary from one domain to another.

## 5.1 Weight Assignment Technique for Quality – Case Study

To establish these facts in real practice, we conducted a survey on software professionals working on e-commerce projects by using component-based technologies in 9 multi-national companies. These professionals have varied experience (5-12 years) ranging from senior software developers to the project managers. All the professionals involved have completed at least three projects on these technologies with varied responsibilities. Survey form consists of all the quality characteristics and sub-characteristics proposed in our model. Professionals were requested to give their preferences on these characteristics and sub-characteristics ranging from *Never Required* - 0 to *Always Required* - 4, while keeping in mind a particular application domain.

Responses received from these professionals are analyzed through Analytical Hierarchy Process (AHP) approach. AHP is a technique that supports decision makers in structuring complex decisions, quantifying intangible factors, and evaluating choices in multi-objective decision situations. It is a comprehensive and rational decision-making framework that provides a powerful methodology for determining relative worth among a set of elements [22]. AHP is especially suitable for complex decisions that involve the comparison of decision elements which are difficult to quantify. Its application has been reported in numerous fields, such as transportation planning, portfolio selection, corporate planning and marketing. It involves:

- Development of relative importance among the attributes using experts' opinion or through exhaustive paired comparison analysis,
- Developing through an algorithm a weightage for each of the attributes,
- Performing similar analysis for the alternative solution strategies for each of the attributes, and

d) Developing a single overall score for each of the alternate solution strategies.

The survey is conducted on 36 professionals and analyzed in three phases by taking first 10 then 25 and finally total 36 persons. After each phase, we analyzed the response by using AHP and we found a maximum of 3% variation in the results, which indicates towards the sufficiency of sample data and shows that there is no much difference in the result if we increase the number of respondents. MS-Excel is used to process the generated data. The Table 4 shows the weight values for each main characteristic in the range from 0 to 1. The sum of all weight values is 1.

Functionality	Reliability	Usability	Efficiency	Maintainability	Portability
0.205	0.184	0.202	0.16	0.139	0.11

Table 4: Weight Values assigned to quality characteristics

This table shows that in eCommerce domain, functionality and usability are two most important characteristics, while workers gave least preference to portability. However, as a characteristic, this lowest value can not be ignored and has to be considered while evaluating the overall quality of the component based system.

Further, weights of sub-characteristics under these characteristics are decided by AHP analysis. The weight value of a characteristic is distributed among its sub-characteristics. These weight values are used to calculate the measured value of a quality characteristic from the measured values of its sub-characteristics. The sum of the weight values of all the sub-characteristics under a characteristic is equal to the weight value of that parent characteristic. Similarly, the sum of all sub-characteristics, irrespective of parent characteristics, is again equal to 1. The weight values of the sub-characteristics are shown in Table 5.

It may be noted from this table that the weight value for functionality is the highest among all, but its sub-characteristics have very less weight values, as compared to other sub-characteristics. Similarly, though efficiency and portability have very less weights, but their constituents are assigned quite large weights as

compared to others. The reason is obvious. Here number of sub-characteristics is not same for all characteristics. Functionality divides its weight value among its seven sub-characteristics, while portability and efficiency divide among three. Therefore, while choosing the sub-

Characteristic	Sub-characteristics	Weight Values	Sum	Grand Total
Functionality	Suitability	0.037	0.205	1.0
	Accuracy	0.043		
	Interoperability	0.028		
	Security	0.039		
	Compliance	0.026		
	Reusability	0.032		
Reliability	Maturity	0.067	0.184	
	Fault Tolerance	0.060		
	Recoverability	0.057		
Efficiency	Time Behavior	0.056	0.16	
	Resource Behavior	0.052		
	Scalability	0.052		
Usability	Understandability	0.057	0.202	
	Learnability	0.048		
	Operability	0.049		
	Complexity	0.048		
Maintainability	Stability	0.022	0.139	
	Testability	0.032		
	Changeability	0.033		
	Trackability	0.026		
	Flexibility	0.026		
Portability	Adaptability	0.043	0.11	
	Installability	0.037		
	Replaceability	0.030		
				1.0

Table 5: Weight Values assigned to quality sub-characteristics

characteristics, the weight of their parent characteristics should also be considered.

The weight values obtained through the process mentioned above can help a developer to select those quality characteristics and sub-characteristics, which are important and relevant as per their quality requirement in that domain. Through this weighting technique, developers can be restricted not to over design the system in favor of a particular characteristic. It will control the development time and financial tradeoff also. But if we choose only selected characteristics and sub-characteristics, the sum of these will not be 1, as we are excluding the weight values of not-so-important characteristics/sub-characteristics. In this case, we can normalize the weight values of selected characteristics/sub-characteristics relatively to make the sum equal to 1. For example, if only suitability, accuracy, security and reusability have to be selected under the functionality characteristic, which have the weight values 0.033, 0.038, 0.035 and 0.028. Other attributes like interoperability, compliance and compatibility are not considered to be important (though these have been assigned certain weight values). We need to normalize these weight values to the selected

sub-characteristics so that the sum of all these selected sub-characteristics will be equal to 0.205 (weight value for functionality characteristic shown in Table 5). The following formula can be used to normalize the weight value of sub-characteristic say  $SC_i$ , which is under characteristics C:

$$\text{Normalized Weight Value } SC_i = \frac{\text{Weight value of the } C}{\sum \text{weight values of selected sub-characteristics}} * \text{weight value of } SC_i$$

The normalized weight values in this case will be .05, 0.058, 0.054 and 0.043. Similarly, we can get the normalized weight values of all the sub-characteristics chosen for a particular application for a domain.

## 5.2 Evaluation of quality as a single variable

Quality is the composition of all its required characteristics and sub-characteristics. To measure it as a single variable, one must include the contribution of each constituent and sub-constituent. The objective of developer must be to achieve all the quality characteristics and sub-characteristics in a best possible way. However, in some cases, it may be possible that if we increase one aspect, other will automatically be decreased. For example, we try to increase the reusability or portability, complexity will be increased. Similarly, to implement more security aspects will reduce the performance. Therefore, efforts need to be made to establish an accepted balance among all these. Following formula is used to evaluate the quality of the system as a whole [20]:

$$Q = w_f F + w_r R + w_u U + w_e E + w_m M + w_p P$$

where  $w_f$ ,  $w_r$ ,  $w_u$ ,  $w_e$ ,  $w_m$  and  $w_p$  are the weight values for the quality characteristics- functionality (F), reliability (R), usability (U), efficiency (E), maintainability (M) and portability (P) respectively. Further, the characteristics are divided into sub-characteristics, the values of F, R, U, E, M and P can be measured by using their constituents like-

$$F = w_s S + w_a A + w_i I + w_{se} Se + w_c C + w_{co} Co + w_{re} Re$$

$$R = w_{ma} Ma + w_{ft} Ft + w_{rec} Rec$$

where S: Suitability, A: Accuracy, I: Interoperability, Se: Security, C: Compliance, Co: Compatibility, Re: Reusability, Ma: Maturity, Ft: Fault Tolerance and Rec: Recoverability. Also, the weight value symbols are used as a multiplier along with the sub-characteristics (like  $w_s$  is the weight value of S). Other characteristics can also be expressed in a similar way.

Now, to evaluate the individual sub-characteristics, we need to use the related metric. But here units of all the metrics may not necessarily be same. Like, some may be measured in numbers; others may be in ratio or just presence. Therefore, we need to normalize these values

in the range of 0 to 1, so that all can be fit under a unique scale. Within the normalized range, the highest value for a metric is 1 and is the maximum achievable level.

## 5.3 Experimental Evaluation of Quality on Industry Software Project

To evaluate the quality as a whole, we conducted an experiment on a software project from the manufacturing domain, which was to be developed in Java and related technologies. A COTS component to be used in the application was selected and its quality was evaluated. After a preliminary consideration, the following characteristics and sub-characteristics were specified by the client for the evaluation:

Functionality	Reliability	Efficiency	Usability	Maintainability
Suitability	Maturity	Scalability	Understandability	Changeability
Accuracy		Resource Behavior	Learnability	Testability
Security				Trackability
Reusability				

Table 6: Selected quality attributes for evaluation

Here portability and its sub-characteristics are not selected, as the application is intended only for a fixed platform. Based on these selected characteristics and sub-characteristics, the normalized weight values are obtained as given in the Table 7 and 8.

Functionality	Reliability	Usability	Efficiency	Maintainability
0.23	0.207	0.227	0.18	0.156

Table 7: Normalized Weight Values assigned for selected quality characteristics

Characteristic	Sub-characteristics	Weight Values	Sum	Grand Total
Functionality	Suitability	0.057	0.230	1.00
	Accuracy	0.066		
	Security	0.060		
	Reusability	0.047		
Reliability	Maturity	0.207	0.207	
Efficiency	Resource Behavior	0.18	0.180	
Usability	Understandability	0.123	0.227	
	Learnability	0.104		
Maintainability	Testability	0.055	0.156	
	Changeability	0.056		
	Trackability	0.045		

Table 8: Normalized Weight Values assigned to selected quality sub-characteristics

Table 9 shows the set of metrics collected to measure these characteristics and sub-characteristics. Out of these selected 11 sub-characteristics, 4 (marked with \*) need to be normalized on 0 to 1 scale.



Parameter	Metric
Suitability	1-(No. of operations not suitable / Total number of operations provided)
Accuracy	Number of operations having required accuracy/ Total number of operations
Security	No. of access controllability provided / Total no. of access controllability required
Reusability	Number of customizable properties / total number of properties
Maturity*	No. of versions released so far for the same component
Resource Behavior	1- (%CPU usage for the execution of the component/100)
Understandability*	Documentation, Help System, Training provided
Learnability	No. of observable properties/Total number of properties
Changeability	Number of customizable properties /total number of properties
Testability*	Sufficient number of test cases provided
Trackability*	Functional and Behavioral tracking system provided for easy maintenance

Table 9: Set of metrics to be used for evaluation

For Maturity, we assume that a component having its first version has maturity 0, second released version 0.25, third released versions 0.5, forth version 0.75 and five or more versions means the mature enough component and it may have maturity value 1.0. Understandability can be measured on the basis of documentation; help system and training provided to the users for the target component and 0.33 each can be assigned for these three constituents respectively. Testability is based on the sufficient number of test cases, so that component can be tested properly. We can measure it by giving values 0, 0.5 and 1 for no test cases provided, provided but not adequate and adequate number of test cases provided respectively. Lastly, trackability can be measured on the basis of tracking facility provided by the component developer so that it can help in future maintenance activities. Again, 0, 0.5 and 1 can be assigned for no tracking provided at all, only functional or behavioral tracking provided, and both functional and behavioral tracking provided respectively.

Now all the sub-characteristics selected for the component are normalized on the scale of 0 to 1 and can be applied on the target component. The values obtained for these metrics are shown in Table 10. Result shows that the quality of the target component to be used in the system is 0.583 on a scale of from 0 to 1.

Chs.	Sub- Chs.	Metric (M)	Wt. (w)	$M_i * w_i$	Quality
Funct.	Suitability	1-1/7 = 0.85	0.057	0.048	0.184
	Accuracy	6/6=1	0.066	0.066	
	Security	3/4=0.75	0.060	0.045	
	Reusability	7/13=0.54	0.047	0.025	
Reliab	Maturity*	0.25	0.207	0.052	0.052

Effic.	Resource Behavior	0.89	0.180	0.160	0.160
Usab.	Understan*	0.66	0.123	0.081	0.129
	Learnability	6/13=0.46	0.104	0.048	
Maint.	Changeab.	7/13=0.54	0.056	0.030	0.058
	Testability*	0.5	0.055	0.028	
	Trackability*	0	0.045	0	
Total					0.583

Table 10: Final value of quality

## 6.0 Discussion

Proposed methodology can be used to estimate the quality of any component before using it in the final system. Also, it may be used to estimate the efforts required to achieve a required value of any characteristic. Like in the above experimentation, the score of the functionality is 0.184 out of 0.205. To achieve a minimum accepted level of functionality for example say 0.19, we need to increase the value of some of its constituent parameters, without affecting others. We can achieve it by putting extra efforts in security by providing one more controllability aspect, so that the metric value for security now is 1.0 and final value of functionality will become more than the minimum required value. However, it may be possible that if we increase security aspect, it will decrease other aspects like complexity or performance. Therefore, we need to have a balance between these values so that the component can be used in the final system. Further, to achieve a particular level of any quality aspect, developers must employ certain techniques, methodologies, tools and processes, each of which has associated costs and benefits. These costs and benefits need be considered while implementing the requirement analysis.

## 7.0 Conclusion

This paper presents a survey of various quality models proposed so far for non-component and component based systems. ISO 9126 is found to be more promising model among all. Present work 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 component context. As, not all characteristics, are required while developing a quality system, we need to select only those characteristics and sub-characteristics which may be of prime importance for that application. For that purpose, a weight assignment technique is used by using the Analytical Hierarchical Process (AHP) concept. We have demonstrated the use of this technique in an experiment by taking a real-life example and evaluated the overall quality of the component. This quality can be used to compare and select the best suitable component as per all desired quality characteristics.

## 8.0 References

- [1] IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990.
- [2] Khosravi, K., Gueheneuc, Y.G., "A Quality Model for Design Patterns", 2004, Online at: <http://www.yanngael.gueheneuc.net/Work/Tutoring/Documents/041021+Kashayar+Khosravi+Technical+Report.doc.pdf>.
- [3] ISO 9126, "Information Technology – Product Quality - Part1: Quality Model", International Standard ISO/IEC 9126, International Standard Organization, June, 2001.
- [4] Losavio, Francisca, Chirinos, Ledis, Pérez, Maria A., "Quality Models to Design Software Architecture", Journal of Object Technology, Vol. 1, Iss. 4, 2002
- [5] McCall, J. A., Richards, P. K., & Walters, G. F., "Factors in Software Quality", Griffiths Air Force Base, N.Y. Rome Air Development Center Air Force Systems Command, 1977.
- [6] Cote, M., A. Suryn, W. Georgiadou, E., "Software Quality Model Requirements for Software Quality Engineering", 14<sup>th</sup> International Conference on Software Quality Management, 2006, 31-50.
- [7] Boehm, B. W., Brown, J. R., Lipow, M. L., Quantitative Evaluation of Software Quality. Proceedings of the 2nd International Conference on Software Engineering, San Francisco, California, United States, 1976, 592-605, IEEE Computer Society Press.
- [8] Maryoly, O., M.A. Perez and T. Rojas, "A Systemic Quality Model For Evaluating Software Products" 2002, available at <http://www.lisi.usb.ve/publicaciones>.
- [9] Dromey, R. G., A model for software product quality. IEEE Transactions on Software Engineering 21, 1995, 146-162.
- [10] Adnan Rawashdeh, Bassem Matakah, "A New Software Quality Model for Evaluating COTS Components", Journal of Computer Science, 2006, Vol. 2 Iss. 4, 373-381.
- [11] I. Crnkovic, M. Larsson, and O. Preiss, "Concerning Predictability in Dependable Component-Based Systems", Classification of Quality Attributes, Architecting Dependable Systems III, 257-278, LNCS 3549, 2005.
- [12] M. Bertoa, A. Vallecillo, "Quality Attributes for COTS Components", In the Proceedings of the 6th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE), Spain, 2002.
- [13] Preiss, O., Weqmann, A., Wong, J., "On Quality Attribute Based Software Engineering", Proceeding of 27th EuroMicro Conference, 2001, 114-121.
- [14] S. Sedigh-Ali, A. Ghafoor, and R. A. Paul, "Software Engineering Metrics for COTS-based Systems". IEEE Computer, 2001, 34(5): 44-50.
- [15] Bertoa, M. and Vallecillo, A. "Usability metrics for software components". Proceedings of Quantitative Approaches in Object-Oriented Software Engineering QAOOSE 2004, Oslo.
- [16] Jon Arvid Borretzen, "The Impact of Component Based Development on Software Quality Attributes", available at <http://www.idi.ntnu.no/emner/dt8100/Essay2005/Boerretzen.pdf>
- [17] Alexandre Alvaro, duardo Santana de Almeida, Silvio Romero de Lemos Meira, "Quality Attributes for a Component Quality Model", Proceeding of 10<sup>th</sup> International Workshop on Component Oriented Programming (WCOP), Glasgow, Scotland, 2005.
- [18] Sharma Arun, Kumar Rajesh, Grover P S, "Empirical Evaluation of Complexity Metric for Software Components", International Journal of Software Engineering and Knowledge Engineering, Vol 19, Iss. 5, to be published in Aug. 2009.
- [19] Mark R Vigder, Anatol W. Kark, "Maintaining COTS-Based Systems: Start with the Design", in Fifth International Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems, 2006, 8–13.
- [20] Grover P S, R Kumar, A Sharma, "Few Useful Considerations for Maintaining Software Components and Component-Based Systems", ACM SIGSOFT Software Engineering Notes, 2007, Vol. 32, Iss. (5), 1-5.
- [21] Voas, Jeffrey, Agresti, William W. (2004): Software Quality from a Behavioral Perspective. IEEE Computer Society. IT Pro., July-August 2004.
- [22] Jayaswal, B. K., Patton, Peter, C., Forman, Ernest, H., "The Analytic Hierarchy Process (AHP) in Software Development", Prentice Hall, 2007.