# Journal of Software Engineering

ISSN 1819-4311

Academic Journals Inc.

www.academicjournals.com

Journal of Software Engineering 4 (1): 1-15, 2010
ISSN 1819-4311


# Q' FACTO 10-A Commercial Off-the-Shelf Component Quality Model Proposal

[1]S. Kalaimagal and [2]R. Srinivasan
[1]Department of CSE, SRM Valliammai Engineering College, Chennai-603203, India
[2]Department of CSE, B.S. Abdur Rahman University, Chennai-600048, India

**Abstract:** This study proposes a quality model that can be used for the evaluation of COTS components quality by organizations. The proposed quality model is primarily based on the ISO 9126 model. Appropriate quality factors, criteria and measures have been identified in the model. Suitable weights are fixed to the different quality factors based on their significance level with respect to COTS component quality. The use of the model is illustrated with a flowchart. A study to demonstrate the use of the model has been presented. The experimental results obtained while validating the model have been presented. The results show that the model has been successful in evaluating the quality of COTS components within specific domains. The next step is the validation of the model on a larger scale to see if it can be applied to generalized domains.

**Key words:** COTS quality factors, COTS quality sub factors, COTS quality measures, ISO9126

## INTRODUCTION

Modern software systems have become more complex. The development process for such complex software also becomes tedious and uncontrollable resulting in high development costs, low productivity and unmanageable software quality (Pour, 1999). Consequently, there was a demand for a new, efficient and cost effective software development paradigm. Component Based Software Engineering (CBSE) was developed as an attempt to meet the challenges of the above paradigm. The component based software development approach is based on the idea of developing software systems by selecting appropriate off-the-shelf COTS components and then assembling them in a well defined architecture. However, organizations that aim to construct software by integrating components (rather than developing software from scratch) will not be able to meet their objectives if they cannot find sufficient number of components and component versions that certify certain functional and quality requirements (Alvaro *et al*., 2005). This means that without a high quality benchmark and systematic certification process, component usage may lead to catastrophic results. As an illustration, we cite the failure of the Arianne Rocket, which is one of the most widely, discussed cases in component research literature (Jezequel and Mager, 1997).

One of the ways that can be used to assure the quality of components is to have them evaluated with the help of a quality model. Most of the existing quality models like ISO9126 (ISO/IEC JTC1, 1999), Dormey (1995) quality model and McCalls quality model describe only the quality characteristics of general software. Further, for building a quality model for COTS components, we are faced with the problem of lack of information provided by COTS

**Corresponding Author:** Sivamuni Kalaimagal, No. 1, Judge Colony, 3rd Street, Tambaram Sanatorium, Chennai-600047, India

component vendors. A visit to the websites of major component vendors like Component Source (www.componentsource.com), Flashline (www.flashline.com) will testify the above statement (Bertoa and Valecillo, 2002). A few quality models have already been proposed for evaluating the quality of COTS components. However, they fail to include important COTS characteristics like reusability and testability.

In order to address many of these issues, this study presents a quality model that has been specifically designed for the evaluation of COTS components. We have built this model as a refinement of existing approaches by pruning many of the attributes that are not directly applicable to the quality domain of COTS components and also by proposing a set of relevant quality attributes in addition to the already proposed ones.

## PROPOSED MODEL

### Assumptions

This quality model has been designed based on the following assumptions:

- The model is meant only for commercial COTS components for which the source code is not available. Any reference to components in the text means commercial COTS components
- The COTS components will have to be quality tested based only on the functionality and market requirement for which it was designed. As an example, if a component is designed for generating charts, it will be tested only on its prime functionality and not on any other functionality, say, as a text editor
- This model is intended only for component end users and third party organizations to evaluate the quality of COTS components. This model has not designed for component developers
- The model is primarily based on the ISO 9126 model

### Outline of Proposed Model

The quality factors, quality criteria and quality measures of the proposed model are outlined below:

### Quality Factor 1-Maintainability

Maintainability is the effort required to replace a COTS component with the corrected version and to migrate an existing, software component from a current component based software system to a new version of the system (Gao *et al*., 2002).

### Quality Criteria (1, 1)
### Stability

The attributes of the COTS component that relate to the risk of unexpected modifications. The greater the stability of the COTS components, the lesser maintenance required (Rawashdeh and Matalkah, 2006).

### Quality Measure (1, 1, 1)
### Component Stability

We propose a COTS Component Stability (CCS) metric that provides an indication of the ability of a software product (based on changes that occur for each version of the software component) based on the Software Maturity Index metric proposed in IEEE std.982.1-1988.

$$CS = [M(t)-(F(a)+F(c)+F(d)]/ M(t)$$

Where:
M (t) = The No.of interfaces provided in the current version
F (a) = The No. of interfaces in the current version that have been changed
F (a) = The No. of interfaces in the current version that have been added
F (d) = The No. of interfaces from the preceding release that were deleted in the current release

As the CS approaches 1.0, we can say that the software component has begun to stabilize.

**Quality Criteria (1, 2)**
**Ease of Migration**

The attributes of the COTS component that relate to the effort required to migrate an existing, software component from a current component based software system to a new version of the system.

**Quality Measure (1, 2, 1)**
**Migration Ease level**

A level measure is used with five levels: 1 for very easy (1.0), 2 for easy (0.8), 3 for slightly difficult (0.6), 4 for very difficult (0.4) and 5 for not at all possible (0.2). The figures in the brackets indicate the values assigned for each level when calculating the overall quality.

**Quality Criteria (1, 3)**
**Replaceability**

The attributes of the COTS component that relate to the effort required to replace a COTS component with the next version (Smith, 2005).

**Quality Measure (1, 3, 1)**
**Replacement Ease level**

A level measure is used with five levels: 1 for very easy to replace (1.0), 2 for easy to replace (0.8), 3 for slightly difficult to replace (0.6), 4 for very difficult to replace (0.4) and 5 for not at all possible to replace (0.2). The figures in the brackets indicate the values assigned for each level when calculating the overall quality.

**Quality Factor 2-Testability**

The testability of software components is one of the most important factors determining the quality of components. Building components with good testability simplifies test operations, reduces test costs and increases software quality. This is more important in the case of COTS components because the components are developed and tested in a totally different site from the place of use of the component. The component buyers usually get only the exe file and a few documents with which they have to carry out the integration testing.

**Quality Criteria (2, 1)**
**Test Documentation**

The attributes of the COTS component that demonstrate whether proper documentation is available for the testing process. This includes the presence of a well-documented test suite, user's manual, well-detailed test cases, description of the testing environment and also the milestones to be encountered in the testing process.

**Quality Measure (2, 1, 1)**

Test Suit documentation measures if whether proper documentation has been provided for plans, tests cases, testing milestones, testing environment etc.

Test Suit Documentation = (1 if present| 0 if absent)

**Quality Measure (2, 1, 2)**

Proofs of Previous Tests measures if the history of previous tests has been provided which demonstrate how successful or unsuccessful the tests were.

Proofs of previous tests = (1 if present| 0 if absent)

**Quality Criteria (2, 2)**

**Component Controllability**

The attributes of the COTS component that indicate how easy is it to control the program or component on its inputs/outputs, operations and behavior. Component users look at the controllability of a COTS component in three aspects: (1) behavior control, (2) feature customization and (3) installation and deployment. Controllability is important because it directly affects testability. Testers and customers expect components to provide a set of control functions in software so that they can use them to check and monitor diverse component behaviors according to their needs.

**Quality Measure (2, 2, 1)**

**Component Execution Control**

This attribute describes whether the component allows the user to control its entire execution from beginning to end. It means that it is possible for the user to start, stop and rerun the execution of the component according to their wish.

Component execution control = (1 if present| 0 if absent)

**Quality Measure (2, 2, 2)**

**Component Environment Control**

This attribute describes whether the component allows the user to control its interaction with the environment in which it is executing.

Component environment control = (1 if present| 0 if absent)

**Quality Measure (2, 2, 3)**

**Component Function Feature Control**

This attribute tell us whether the component allows the user to switch control between the different functionalities offered by the component while the component is executing.

Component function feature control = (1 if present| 0 if absent)

**Quality Criteria (2, 3)**

**Traceability**

The attributes of the COTS component that refers to the extent of its built in capacity of tracking the status of component attributes and component behavior. In the real world, engineers have trouble carrying out operation and state traces in COTS components due to

the inaccessibility of the source code. A few research efforts that allow COTS components to be tracked by adding a program tracking mechanism have been attempted (Gao *et al*., 2002). However, they are not so successful, due to the lack of standardized component trace formats and trace mechanisms. Performance trace and error trace can be made possible with the help of log files.

**Quality Measure (2, 3, 1)**
**Performance Trace**

This attribute describes whether the component allows the user to perform a performance trace of the component execution.

Performance trace = (1 if present| 0 if absent)

**Quality Measure (2, 3, 2)**
**Error Trace**

This attribute describes whether the component allows the user to perform an error trace of the component execution.

Error trace = (1 if present| 0 if absent)

**Quality Factor 3-Functionality**

This characteristic expresses the ability of a component to provide the required services and functions that meet stated and implied needs when the component is used under specified conditions.

**Quality Criteria (3, 1)**
**Self-Contained**

One of the characteristic properties of a component is that it should be self-contained (Szysperski *et al*., 2002). It is an intrinsic property of the component and it means that the component is encapsulated with well-defined interfaces and can be executed independently with minimal outside support. He also says that well-defined interfaces have contracts that are described by the presence of preconditions and post conditions (Szysperski *et al*., 2002).

**Quality Measure (3, 1, 1)**

Presence of preconditions and postconditions indicates if each interface in the component has well defined preconditions and postconditions together with their which determine what the component provides and requires.

Presence of preconditions and postconditions = (1 if present| 0 if absent)

**Quality Measure (3, 1, 2)**

Modularity measures the degree of independence of the component, which is the degree of functional cohesion.

$$\text{Modularity} = \frac{\text{Functions provided by the component itself without external support}}{\text{Total functions provided by the component}}$$

**Quality Criteria (3, 2)**
**Compliance**

The attributes of the component that make the software adhere to application related standards or conventions or regulations in laws and similar prescriptions.

**Quality Measure (3, 2, 1)**

Presence of Standardization indicates if the component conforms to international standards.

Presence of standardization = (1 if present| 0 if absent)

**Quality Measure (3, 2, 2)**

Presence of Certification indicates if the component has been certified by an internal or external organization

Presence of certification = (1 if present| 0 if absent)

**Quality Criteria (3, 3)**
**Accuracy**

Accuracy is the ability of the component to execute efficiently and deliver the expected results.

**Quality Measure (3, 3, 1)**

Computational Accuracy evaluates the number of accurate results returned by the component operations according to the user specifications.

$$\text{Computational accuracy} = \frac{\text{Accurate results}}{\text{Total results}}$$

**Quality Factor 4-Efficiency**

It is the capability of a component to provide appropriate performance, relative to the amount of resources used under stated conditions. Efficiency is usually measured in terms of time and resource usage. Hence, efficiency measurement of a software component will be just the same as for any other software product.

**Quality Criteria (4, 1)**

Resource Behavior: The attributes of the component that relate to the amount of resources used and the duration of each use in performing its functions.

**Quality Measure (4, 1, 1)**

Disk Capacity measures the attribute specifies the disk capacity used by the component.

$$\text{Disk capacity} = \frac{\text{Amount of disk capacity used}}{\text{Total disk capacity available}}$$

**Quality Criteria (4, 2)**
**Time Behavior**

The attributes of the component that relate to response and processing times and throughput rates in performing its functions.

**Quality Measure (4, 2, 1)**

Response time measures the time taken since a request is received until a response has been sent and is indicated using a time metric.

Response time = Response time of request- Time at which request is received

**Quality Factor 5-Reliability**

Reliability is the capability of the component to maintain a specified level of performance when used in stated conditions in a stated period of time. Reliability can be assessed by measuring the frequency and severity of failure, the accuracy of output results, the mean time between failures and the ability to recover from failure and the predictability of the program (Rawashdeh and Matalkah, 2006).

**Quality Criteria (5, 1)**
**Recoverability**

The attributes of the software that relate to the capability to reestablish its level of performance and recover the data directly affected in case of failure and the time and effort needed for it.

**Quality Measure (5, 1, 1)**

Persistence denotes the ability of the component to store its state in a persistent manner for later recovery.

Persistence = (1 if present| 0 if absent)

**Quality Measure (5, 1, 2)**

Presence of Fault Tolerant Mechanism indicates whether fault tolerant mechanisms have been implemented in the component.

Presence of fault tolerant mechanism = {1 if present| 0 if absent}

**Quality Factor 6-Usability**

Usability is the capability of the component to be understood, learnt and used under specified conditions. Usability is the effort required to learn, operate, prepare input and interpret out of the program. In the case of COTS and reusable COTS components only the component users can measure this characteristic because the component developers are generally third party vendors who have no connection whatsoever to the component users.

**Quality Criteria (6, 1)**
**Learnability**

The attributes of a software component that relate to the user's efforts for learning its application (for example, operation control, input and output).

**Quality Measure (6, 1, 1)**

Help system measures the quality of the Help system provided with the component for discovering and understanding its services in terms of its completeness, clarity and usefulness. A level measure is used with five levels: 1 for excellent (1.0), 2 for very good (0.8), 3 for good (0.6), 4 for average (0.4) and 5 for a poor Help system (0.2). The figures in the brackets indicate the values assigned for each level when calculating the overall quality.

**Quality Measure (6, 1, 2)**

Training indicates whether training courses are available for the component and if available whether information about the training is provided.

Training = (1 if present| 0 if absent)

**Quality Measure (6, 1, 3)**

Demonstration Coverage measures the percentage of the component services that are shown in the demo compared to the total number of provided services and interfaces.

Demonstration coverage = (1 if present| 0 if absent)

**Quality Factor 7-Security**

The security of the component relates to its ability to prevent unauthorized access, whether accidental or deliberate to programs or data.

**Quality Criteria (7, 1)**
**Access Control**

This attribute has also been slightly refined and added to the model. This attribute indicates how the component is able to control access to its provided interfaces. Example can be a component that provides interfaces with the functionality to identify or authenticate users.

**Quality Measure (7, 1, 1)**

Access controllability measures whether access control mechanisms are implemented or not and if implemented followed by a description of the mechanism used.

Access controllability = (1 if present| 0 if absent)

**Quality Factor 8-Portability**

Portability is the ability of a component to be transferred from one environment to another. Portability is an intrinsic property to the nature of components, which are in principle designed and developed to be reused in different environments (Bertoa and Valecillo, 2002).

**Quality Criteria (8, 1)**
**Installability**

The attributes of the component that relate to the effort needed to install the component in a specified environment.

**Quality Measure (8, 1, 1)**

Installability Documentation indicates if the component developer has provided the component user proper documentation outlining the installation steps.

Installability documentation = (1 if present| 0 if absent)

**Quality Measure (8, 1, 2)**

Installability Complexity is a measure of how complex it is to carry out the installation of the component. A level measure is used with five levels: 1 for very easy (1.0), 2 for easy

(0.8), 3 for slightly difficult (0.6), 4 for very difficult (0.4) and 5 for a poor installation system (0.2). The figures in the brackets indicate the values assigned for each level when calculating the overall quality.

**Quality Criteria (8, 2)**
**Deployability**

The attributes of the component that relate to the effort needed to deploy the component in a specified environment.

**Quality Measure (8, 2, 1)**

Deployability documentation indicates if the component developer has provided the component user proper documentation outlining the deployment steps.

Deployability documentation = (1 if present| 0 if absent)

**Quality Measure (8, 2, 2)**

Deployability complexity is a measure of how complex it is to carry out the deployment of the component. A level measure is used with five levels: 1 for very easy (1.0), 2 for easy (0.8), 3 for slightly difficult (0.6), 4 for very difficult (0.4) and 5 for a poor deployment system (0.2). The figures in the brackets indicate the values assigned for each level when calculating the overall quality.

**Quality Criteria (8, 3)**
**Adaptability**

The attributes of the software that relate to on the opportunity for its adaptation to different specified environments without applying other actions or means than those provided for the component to be considered.

**Quality Measure (8, 3, 1)**

Mobility indicates whether the component can be deployed in any other container other than the one it was transferred in.

Mobility = (1 if present| 0 if absent)

**Quality Factor 9-Interoperability**

The attributes of the component that reflect its ability to interact with other systems whether they are composed of components or not.

**Quality Criteria (9, 1)**
**Compatibility**

This is the attribute that tells us whether the data format or versions of the component is compatible with that of interacting components.

**Quality Measure (9, 1, 1)**

Data Compatibility measures whether the format of the data used handled by the component is compliant with international standards or conventions so that the component can interact with other components.

Data compatibility = (1 if present| 0 if absent)

**Quality Measure (9, 1, 2)**

Version compatibility indicates if the component is compatible with backward and forward versions of the same component.

Version compatibility = (1 if present| 0 if absent)

**Quality Factor 10-Reusability**

Reusability is important in the development of a component-based system (Jon, 2000). He also says that in the context of component based software engineering; reusability can refer to the ability to reuse existing components to create a more complex system. There is a great demand for reusable components in the component market. Reusability is also a major driving force of the component market (Jon, 2000). There are expectations that a high quality component has to be reusable. Hence, it is becomes necessary to include reusability as a quality factor in our quality model.

**Quality Criteria (10, 1)**

**Generality**

A reusable component has to be generic in nature. Generality is the set of attributes that determine how reusable the component and it's capability of extending specific interaction knowledge to new situations.

**Quality Measure (10, 1, 1)**

Presence of domain abstraction measures whether the component can be re used across several domains related to the specific functionality that the component offers.

Presence of domain abstraction = (1 if present| 0 if absent)

**Quality Measure (10, 1, 2)**

**History of Reuse**

This attribute indicates if previous uses of the reusable component have been recorded and documented for future reference.

History of reuse = {1 if present| 0 if absent}

**Quality Criteria (10, 2)**

**Hardware/Software Independence**

It would be highly favorable for a reusable component not to depend on any particular hardware or software architecture.

**Quality Measure (10, 2, 1)**

Presence of hardware independence measures if the component is dependent or not dependent on any particular hardware.

Presence of hardware independence = (1 if present| 0 if absent)

**Quality Measure (10, 2, 2)**

Presence of software independence measures if the component is dependent or not dependent on any particular software.

Presence of hardware independence = (1 if present| 0 if absent)

**Quality Criteria (10, 3)**

**Locatability**

The attributes of the component that determine the ease with which the appropriate reusable component can be obtained from a component repository.

**Quality Measures (10, 3, 1)**

Accessibility measures the ease with which the reusable component can be easily located from a component repository. A level measure is used with five levels: 1 for very easy to locate (1.0), 2 for easy to locate (0.8), 3 for slightly difficult to locate (0.6), 4 for very difficult to locate (0.4) and 5 for able to locate only with external support (0.2). The figures in the brackets indicate the values assigned for each level when calculating the overall quality. Figure 1 shows an overall view of the Q'Facto 10 model.

**Quality Calculator**

The model can be used to calculate the overall quality of a COTS component. The following points are to be noted:

- A threshold value will be set for each quality factor depending on the application of the COTS component, which can be decided by a COTS knowledge resource person
- Weights are assigned to each quality factor and the component user may choose these weights

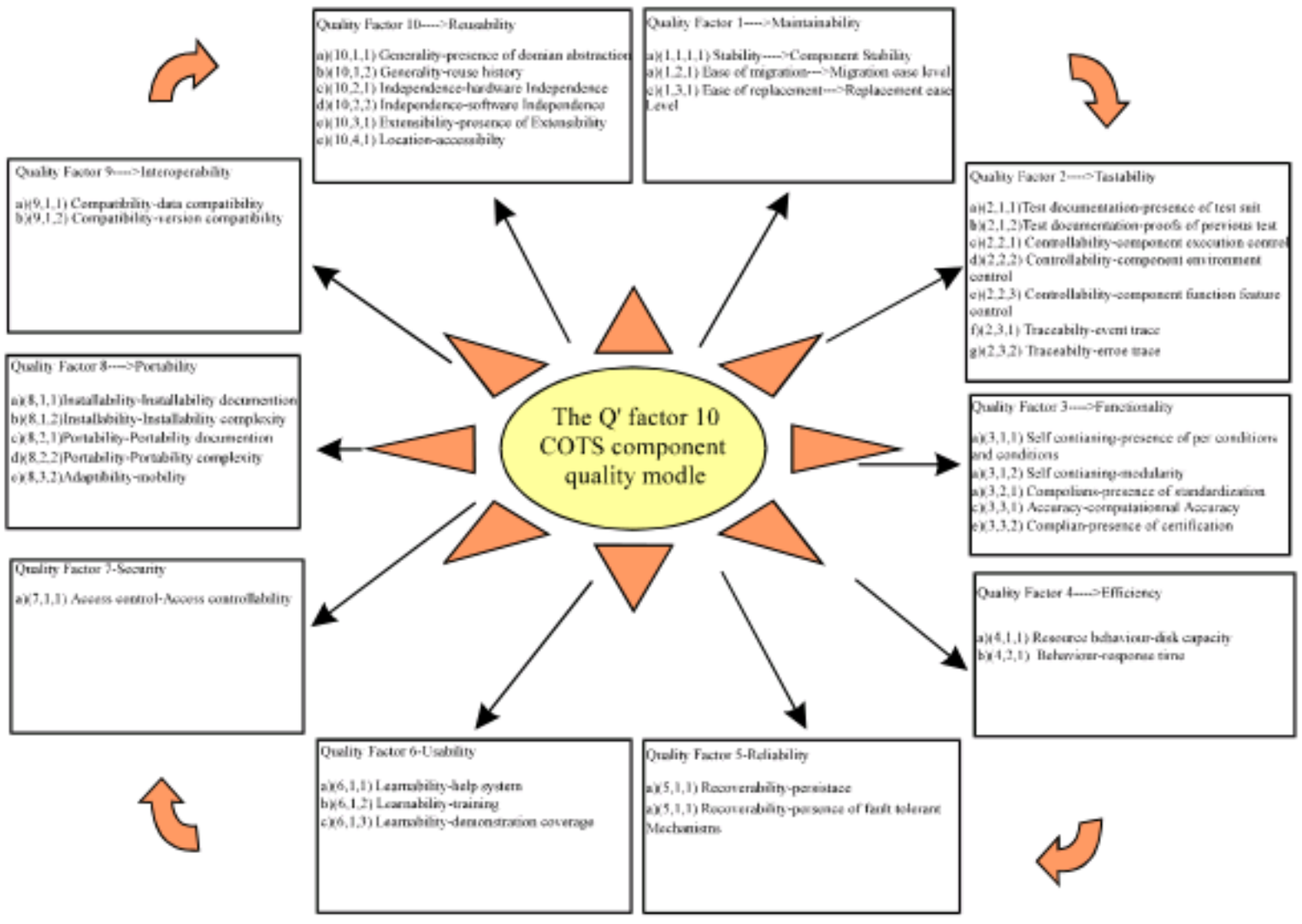


Fig. 1: The Q'Facto10 COTS quality model

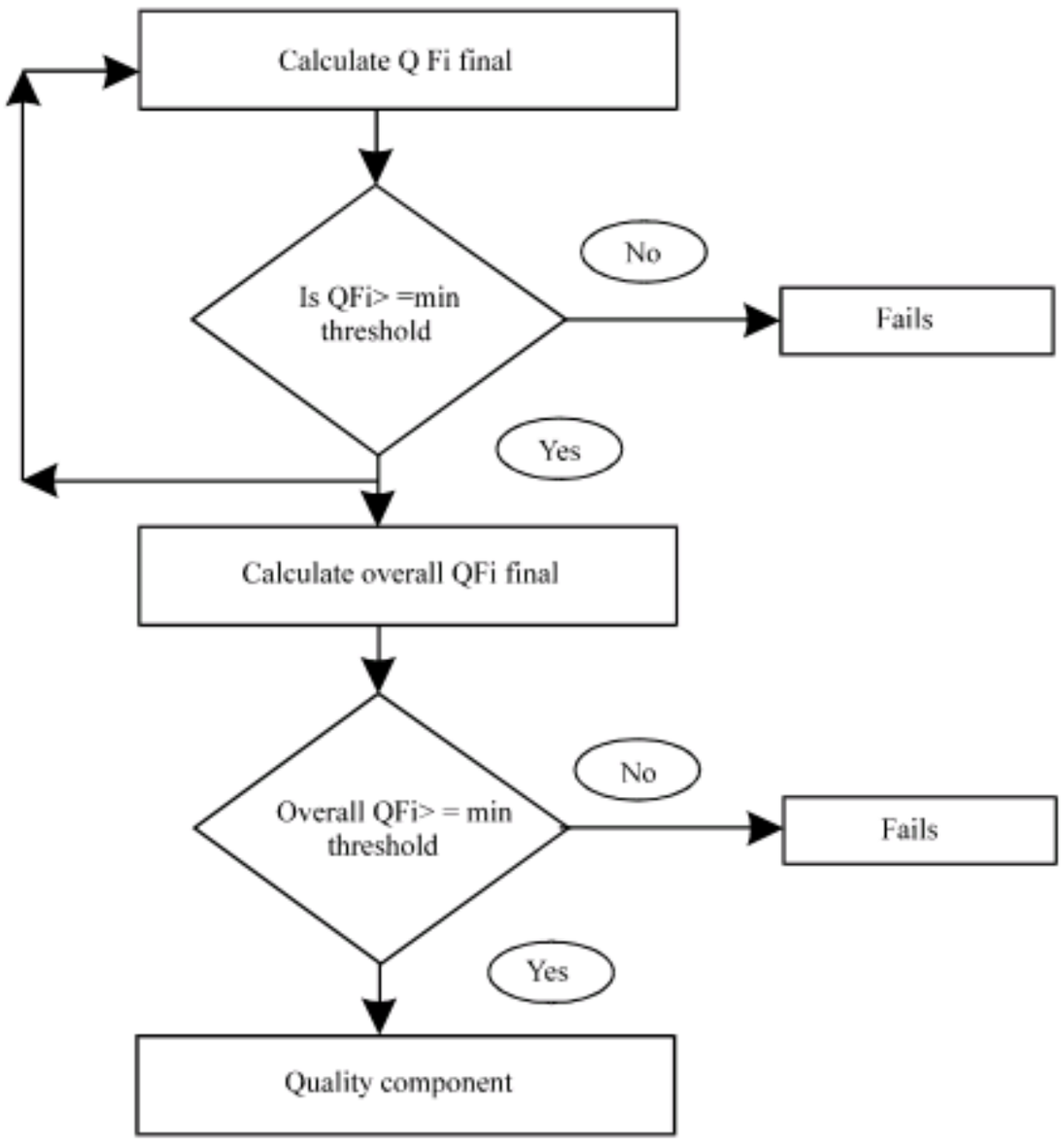


Fig. 2: Quality calculation flow chart

- We consider all the 10 factors important for the overall quality of the component. A COTS component should cross the threshold value of each Quality factor, to certify as a high quality component. If the COTS component fails to cross the minimum value in any quality factor it fails the test for a high quality component

Figure 2 shows the steps to be followed for measuring the quality of the component which can be listed as follows:

The value of each Quality Factor of the component is measured using the formula:

$$QFi = \frac{\sum^{n} i=1\ QM_{i,j,k} * W_{i,j,k}}{N}$$

where, $QM_{i,j,k}$ = kth quality measure of the jth quality criteria belonging to the ith quality factor e.g., ($QM_{1,1,1}$ is the quality measure component stability)

$N_i$ = The total number of quality measures for the Quality Factor QFi (e.g.,: For QF1 the value of N is 3) $W_{i,j,k}$ = weight assigned to Quality measure QMi,j,k

- The value calculated above should be greater than the minimum threshold value set for that particular quality factor. If the value exceeds the threshold value, the next quality factor value i.e. $QF_{i+1}$ will be calculated using the formula above. If the value does not exceed the threshold value, it means that the component fails in that particular quality factor and is classified as a poor component

- Steps 2, 3, 4 should be repeated for each of the remaining quality factors of the model

If the component passes in all the factors then the overall quality of the component can be calculated using the formula:

$$\text{Overall quality} = \frac{\sum_{i=1}^{10} QFi10}{10}$$

- A minimum threshold value will be set for the overall quality factor by the component user. The overall quality factor should cross this threshold value also for it to be certified as a high quality component

## RESULTS

To illustrate the use of the model, we took as examples two component software projects that were downloaded from the internet and given to two different university study groups so that they can analyze and evaluate the model for their projects. The component software projects downloaded from the internet are JGantt and GanttBiz that can be integrated with projects to create gantt charts. Each study group had five members. The results of the investigation have been shown in Table 1. The first column of the table lists out the quality measures, the second column gives the values obtained for the JGantt software for each of the measures and the third colum lists out the values obtained for Gantt project software for each of the measures.

All the measures have been calculated using the metrics explained in the section II. For example, let us consider the calculation of Quality Factor 2, Testability. There was no test suit documentation or error trace log available for the JGantt software .So it has the value of 0 for both the metrics . On the other hand, proofs of previous tests , component function feature control, component environment control were all available and each of the metric has a value of 1. We considered a uniform weight of 1 for each of the metrics. So, the quality measures calculated will be,

- Test suit documentation = value *weight = 0*1 = 0
- Proofs of previous tests = value* weight = 1*1 = 1
- Component execution control = value* weight = 1*1 = 1
- Component environment control = value* weight = 1*1 = 1
- Component function feature control = value* weight = 1*1 = 1
- Performance trace = value* weight = 1*1 = 1
- Error trace = value* weight = 0*1 = 0

Hence, the value of the testability quality factor for JGantt software is (1+1+1+1+1)/5 = 0.71. Similarly, the value of the testability quality factor for Gantt project was 0.71. After calculating individual quality factors, the overall quality factor was calculated using the formula:

$$\frac{\sum_{i=1}^{10} QFi}{10}$$

Overall quality factor of JGantt software = (0.67+0.71+0.78+0.011+1.0+0.66+1.0+1.0+1.0+0.6)/10 = 0.74

Table 1: Component evaluation study results

| Uality measure | JGantt | GanttBiz |
|---|---|---|
| Component stability (Maintainability) | 0.6 | 0.5 |
| Migration ease level (Maintainability) | 0.6 | 1.0 |
| Replacement ease level (Maintainability) | 0.8 | 1.0 |
| **Quality Factor 1 (Maintainability)** | 0.67 | 0.83 |
| Test suit documentation (Testability) | 0 | 0 |
| Proofs of previous tests (Testability) | 1.0 | 0 |
| Component execution control (Testability) | 1.0 | 1.0 |
| Component environment control (Testability) | 1.0 | 1.0 |
| Component function feature Control (Testability) | 1.0 | 1.0 |
| Performance trace (Testability) | 1.0 | 1.0 |
| Error trace (Testability) | 0 | 1.0 |
| **Quality Factor 2 (Testability)** | 0.71 | 0.71 |
| Presence of preconditions and post conditions (Functionality) | 0 | 0 |
| Modularity (Functionality) | 1.0 | 0.888 |
| Presence of standardization (Functionality) | 1.0 | 1.0 |
| Presence of certification (Functionality) | 1.0 | 1.0 |
| Computational accuracy (Functionality) | 0.92 | 0.98 |
| **Quality Factor 3 (Functionality)** | 0.78 | 0.77 |
| Disk capacity (Efficiency) | 0.002 | 0.001 |
| Response time (Efficiency) | 0.02 | 0.03 |
| **Quality Factor 4 (Efficiency)** | 0.011 | 0.015 |
| Persistence (Reliability) | 1.0 | 1.0 |
| Presence of fault tolerant mechanism (Reliability) | 0 | 0 |
| **Quality Factor 5 (Reliability)** | 1.0 | 1.0 |
| Help system (Usability) | 1.0 | 1.0 |
| Training (Usability) | 0 | 0 |
| Demonstration coverage (Usability) | 1.0 | 1.0 |
| **Quality Factor 6 (Usability)** | 0.66 | 0.66 |
| Access controllability (Security) | 1.0 | 1.0 |
| **Quality Factor 7 (Security)** | 1.0 | 1.0 |
| Installability documentation (Portability) | 1.0 | 1.0 |
| Installability complexity (Portability) | 1.0 | 1.0 |
| Deployability documentation (Portability) | 1.0 | 1.0 |
| Deployability complexity (Portability) | 1.0 | 1.0 |
| Mobility (Portability) | 1.0 | 1.0 |
| **Quality Factor 8 (Portability)** | 1.0 | 1.0 |
| Data compatibility (Interoperability) | 1.0 | 1.0 |
| Version compatibility (Interoperability) | 1.0 | 1.0 |
| Quality factor 9 (Interoperability) | 1.0 | 1.0 |
| Presence of domain abstraction (Reusability) | 0..0 | 0.0 |
| History of reuse (Reusability) | 0.0 | 0.0 |
| Presence of hardware independence (Reusability) | 1.0 | 1.0 |
| Presence of software independence (Reusability) | 1.0 | 0.0 |
| Accessibility (Reusability) | 1.0 | 1.0 |
| **Quality Factor 10 (Reusability)** | 0.6 | 0.4 |

Here, we have assigned uniform weights to all the quality factors. The relative weights can be chosen at the discretion of specific user groups. In such a case, the composite quality factor obtained is an indicator of the quality as perceived by a specific user group. Possibly, such an approach can be adopted until a common quality consensus is achieved among heterogeneous user groups.

## CONCLUSIONS

A survey was conducted of the existing models in component literature (Kalaimagal and Rengaramanujam, 2008). Based on the results of the survey, a comprehensive quality model that could be used by user groups for evaluating the quality of COTS components has been proposed in this study. This model is a first step in our research effort in the direction of building a COTS component quality assurance framework.

The model was designed in such a way as to overcome most of the drawbacks of the existing models. Appropriate weights are assigned to the different quality criteria based on their significance level with respect to COTS component quality. An illustration for calculating the quality factors has been provided. Currently, user groups from the software industry have been identified for the real time experimental validation of the model, which will be the next step in our research. This will be followed by the creation of a capability maturity model for components that can be mapped to the proposed COTS quality model.

## ACKNOWLEDGMENTS

The authors record grateful thanks to Dr. Kanniappan VC, Dr. V.M. Periasamy, the Registrar, B.S. Abdur Rahman University and Dr. Balasubramaniam, the Principal,Valliammai Engineering College for the facilities provided. They would also like to express their thanks to Professor Manu Natarajan, Head of the Department of CSE, B.S. Abdur Rahman University for useful discussions.

## REFERENCES

Alvaro, A., A.S. Almeida and S.R.I. Meira, 2005. Quality attributes for a component quality model. Proceedings of 10th International Workshop on Component Oriented Programming (WCOP) in Conjunction with the 19th European Conference on Object Oriented Programming (ECOOP), July 25-29, Glasgow, Scotland, pp: 90-94.

Bertoa, M. and A. Valecillo, 2002. Quality attributes for cots components. Proceedings of the 6th International ECOOP Workshop on Quantitative Approaches in Object Oriented Software Engineering (QAOOSE), June 11-13, Malaga, Spain, pp: 87-90.

Dormey, G.R., 1995. A model for software product quality. IEEE Trans. Software Eng., 21: 146-162.

Gao, J., K. Gupta, S. Gupta and S. Shim, 2002. On building testable software components. Proceedings of the 1st International Conference on COTS-Based Software Systems, Feb. 4-6, Springer-Verlag, London, UK., pp: 108-121.

ISO/IEC JTC1, ISO-14598-1, 1999. Quality and quality characteristics. Technical Report, ISO.

Jezequel, J.M. and B. Meyer, 1997. Design by contract: The lessons of ariane. IEEE Comput., 30: 129-130.

Jon, H., 2000. Component primer. Commun. ACM, 43: 27-30.

Kalaimagal, S. and S. Rengaramanujam, 2008. A retrospective on software component quality models. ACM SIGSOFT Software Eng. Notes, 33: 1-10.

Pour, G., 1999. Software component technologies: JAVA beans and active X. Proceedings of Technology of Object Oriented Languages and Systems, July 28-Aug. 1, Santa Barbara, California, pp: 419-421.

Rawashdeh, A. and B. Matalkah, 2006. A new software quality model for evaluating cots components. J. Comput. Sci., 2: 373-381.

Smith, F.D., 2005. Achieving quality requirements with reused software components-challenges to successful reuse. Proceedings of 2nd International Workshop on Models and Processes for the Evaluation of Off-The-Shelf Components (MPEC), May 15-21, St. Louis, Missourie, USA., pp: 1-3.

Szysperski, C., G. Dominic and M. Stephen, 2002. Component Oriented Programming-Beyond Object Oriented Software. 2nd Edn., Addison Wesley and ACM Press, New York.