CHAPTER 4

THE Q’FACTO 10 QUALITY MODEL

4.1 INTRODUCTION

Component Based Software Engineering is gaining substantial interest in the software community today. This is because a software component offers developers the double-edged sword of shorter life cycles and lesser development costs. However, several technical issues still remain unsolved before the burgeoning software component industry. One such issue is the evaluation of COTS component quality. To evaluate software quality means to perform a systematic investigation of the software capability to meet specified quality requirements. A quality model should be defined to evaluate software quality (Patrick Berander et al 2005). We are proposing a quality model called the Q’Facto10 quality model that can be used to evaluate the quality of COTS components in this chapter.

Most of the existing quality models like ISO 9126 (ISO/IEC 2001), Dromey (1995) and McCall’s quality model (1977) describe only the quality characteristics of general software. Not only that, even to build a quality model exclusively for COTS components, we are faced with the problem of lack of information provided by software component vendors. Visiting 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).

Initially, we designed a quality model that could be used for the evaluation of all kinds of components was built. Then the Q’Facto 10 model
was designed as a refinement of the general component quality model 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 quality attributes relevant to the COTS domain in addition to the already proposed ones.

The methodology used to design the model has been outlined in this chapter. Appropriate quality factors, criteria and measures are identified for the model. The use of the model is illustrated with a flowchart. A study to demonstrate the use of the model has been presented. First, let us have a look at the drawbacks of the existing quality models.

4.2 DRAWBACKS OF EXISTING MODELS

A survey of existing quality models was presented in chapter 2. We found that the quality models had the following common drawbacks:

1. None of the quality models have been designed in a systematic way using a proper methodology, except the model proposed by Rawesdah (2006). All the models are ad hoc variations of the ISO 9126 quality model.

2. A target audience has not been identified for most of the models. Therefore some of the attributes mentioned in the model can be measured only by the developer and not by the end user.

3. Certain high level quality attributes that are important for COTS components like reliability and testability have been ignored.

4. COTS components will be used by a wide range of users and since different users have different perspectives, the measure and importance of each attribute will be different. Therefore it would be advisable to follow a weighted scoring method for
quality evaluation. However, none of the models follow a weighted scoring approach.

5. There is no mention of a method to use the proposed quality models to calculate the quality of the COTS component.

We have tried to overcome the drawbacks mentioned in the design of our proposed Q’Facto 10 quality model. A methodology has been used that was adopted from Dromey (1998). The target audience and requirements of the model have been identified and a weighted scoring method has been used.

The proposed model is based on Alvaro’s quality model (Alvaro et al 2005) and the ISO 9126 model. Although the model proposed is based on the above mentioned two models, some major changes were made in order to develop a consistent model to evaluate software components. One such change is the inclusion of reusability and testability as high level quality factors in our quality model. In this way, we have added characteristics to the model that are relevant to the COTS component context, eliminated characteristics that are not important to evaluate COTS components and changed the names of some characteristics in order to make them more applicable to the COTS component context.

4.3 ASSUMPTIONS OF THE PROPOSED MODEL

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

1. The model is meant only for commercial COTS components for which the source code is not available. We have not included open source COTS components because even though they are available in the market, proper documentation is not available. Hence, any reference to components in the text means commercial COTS components
2. 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.

3. This model is intended only for component end users and third party organizations to evaluate the quality of COTS components. This model has not been designed for component developers.

4. The model is primarily based on the ISO 9126 model (ISO/IEC 2001). This is because most of the existing models are based on the ISO 9126 model since the ISO 9126 model is considered as the most complete software quality model framework proposed so far as discussed in chapter 1.

4.4 TERMINOLOGY

Software is composed of both high and low-level components. Through these components and through the ways they are composed, software exhibits properties that characterize and distinguish it from other artifacts. The words: characteristics, factors and attributes are used as alternatives for what we will refer to here as properties (Dromey 1998). Several different kinds of properties can be distinguished as either tangible (concrete) or abstract (intangible) and either functional or non-functional properties. Tangible properties must be measurable, assessable, calculable or detectible by either manual or automated means.

Some of the properties of software are desirable. These desirable properties are called as qualities or quality attributes. Quality or more specifically, a set of quality attributes is the vehicle through which the different interest groups express their needs of software. A goal directed
approach to building a quality model for software is effective for accommodating and balancing the needs of these different interest groups. The set of desirable properties or quality attributes of software provides an abstract or high-level specification for what we will call software product quality (Dromey 1998).

The quality attributes we associate with software correspond either to “behaviors” or “uses”.

A “behavior” is something that the software itself exhibits when it executes under the influence of a set of inputs software (e.g. reliability and functionality are uses). From a functional perspective, a behavior of a software system can be characterized by the set of responses (including outputs and metrics) the system exhibits through the execution and interaction of sets of its functions in response to one or more sets of inputs.

A “use” is something that different interest groups do with or to software (e.g., portability and maintainability are uses). We may also give uses a functional interpretation.

4.5 PRINCIPLES AND AXIOMS

The model has also been designed using the principles and axioms suggested by Dromey (1998).

Principle 4.1 A behavior can be decomposed and hence defined in terms of subordinate properties which may be described either as behaviors or software characteristics.

Figure 4.1 illustrates this notion schematically. Subordinate behaviors may in turn be hierarchically defined.
As an example, in ISO 9126 (ISO/IEC 2001), reliability can be decomposed into the determinates fault-tolerance, maturity and recoverability.

**Principle 4.2** A use can be decomposed and hence defined in terms of subordinate properties which may be described either as uses or software characteristics.

Figure 4.2 illustrates this notion schematically. Subordinate uses may in turn be hierarchically defined.

As an example, in ISO 9126 (ISO/IEC 2001), usability can be decomposed into the determinates learnability and operability. Let us now state the important axioms of Dromey (1998).
Axiom 4.1 If we admit that software is composed of components then the choice of those components, their tangible intrinsic and contextual properties and the way those components are composed, determines the quality of software.

Axiom 4.2 Software exhibits a set of quality attributes and it exhibits certain observable behaviors and uses that correspond directly to its quality attributes.

Axiom 4.3 Tangible quality-carrying properties of software components contribute to one or more intangible, high-level quality attributes of software.

Axiom 4.4 Associated with each tangible quality carrying property of a component is a verifiable empirical statement that links the property either to a software characteristic, a behavior or a use and then to a high-level quality attribute.

4.6 MODEL FRAMEWORK

The above terminologies, principles and axioms will be used in the model design together with the following assumptions (Dromey 1998):

1. The quality of software may be characterized by a set of high level quality attributes.

2. The quality attributes of software corresponds either to a set of domain dependent behaviours of software or a set of domain independent uses of software.

3. The quality attributes of a quality model should be sufficient to meet the needs of all interest groups associated with the software.
4. Each high level quality attribute of software is characterized by a set of subordinate properties which are behaviors, uses or software characteristics.

5. Each software characteristic is determined or contributed to by a set of tangible properties that we will call quality carrying properties.

6. Quality carrying properties may embody either functionality (e.g. a check to see whether all inputs are within their expected ranges contributes to the design principle of modular protection) or non-functional properties (e.g. identifiers should be self-descriptive).

Our quality model has been designed based on the philosophy of earlier sections.

Figure 4.3 presents the schematic of the proposed quality model.

![Quality Model Schematic](image-url)

**Figure 4.3 Quality Model Schematic**
The quality model comprises ten high level quality factors or characteristics that can either be behaviors or uses. Each high level characteristic will be divided into sub-factors called quality criteria which again can either be a behavior or a use. Each quality criteria in turn will have quality measures that can be used to empirically define the quality criteria as stated in axiom 4.

4.7 MODEL METRICS

One of the objectives that motivated the model construction is that the model should be comprehensive and easy to use by the end users. We have therefore chosen metrics that can be easily used to measure quality. The metrics used in the model are:

1. **Presence:** This metric is used to indicate whether an attribute is present in the component or not. If it exists a 1 is used, else a 0 is used.

2. **Level:** This metric is used to indicate the degree of effort; ability etc. It is usually a subjective measure. It is described by an integer variable that can take any value in the fractional range from 0 to 1. An example can be: 1 for Very Very Easy (1.0), 2 for Very Easy (0.8), 3 for Easy (0.6), 4 for difficult (0.4) and 5 for very difficult (0.2) and 6 for not at all possible (0.0). The figures in the brackets indicate the values assigned for each level when calculating the overall quality.

3. **Ratio:** This is used to describe percentages.

4.8 DESIGN METHODOLOGY

The methodology used in designing the proposed model is outlined in the steps below:
A. **Step 1:** Identify the stakeholders (types of users) or intended audience of the quality model

B. **Step 2:** The model is constructed in a hierarchical top down fashion using the following steps:

**Step 2.1:** Identifying a small set of agreed upon and non intersecting high level quality factors that can describe the basic quality requirements that a software component should satisfy.

**Step 2.2:** Decomposing each high level quality factor into a set of quality sub-factors or quality criteria in a top down fashion

**Step 2.3:** For every quality criterion identified in the above step, determining the way in which it can be measured and represented.

### 4.8.1 Identifying Stakeholders

The term stakeholder is used to refer to the audience who will be using the quality model. The proposed model is specifically targeted to end users who will use the component and not developers. End users can be classified into the following categories:

1. **Component Evaluator:** The component evaluator chooses the appropriate component required for the application from a component repository after evaluating the adaptability of the component to the application.

2. **Component Integrator:** The component integrator integrates the component with the application.

3. **Tester:** It is the responsibility of the tester to test the entire software after integration with the component.
4. **Project Manager:** The project manager of the project team that is going to use the purchased COTS component.

5. **End User:** The end user only needs to know the component functionality and how it can be used via interfaces.

### 4.8.2 Quality Factors, Criteria and Measures

We have identified ten high level quality factors, 21 quality criteria and 35 quality measures for the Q'Facto 10 quality model. Since this model is a frontrunner for our final Q'Facto 12 quality model, we shall describe how the quality factors and criteria and measures were chosen in this chapter and the other details will be provided in the next chapter.

#### 4.8.2.1 Quality factor 1: maintainability

In the ISO 9126 quality model (ISO/IEC 2001), maintainability is the effort needed to make specified modifications. When we talk of COTS components as per our definition in chapter 3 and if we go by the assumption in section 4.2 that we consider only COTS components whose source code is not available then there is no place for any modification. However, it is possible to carry out two kinds of changes with regards to COTS components that are independent of the source code. One change is replacing the current version of the COTS component with a higher version of the same component called as Replacement and another change is transferring the COTS component from a current version of the existing system to a new version of the system called as Migration. It should be easy for the end user to carry out both replacement and migration. We therefore subdivide the quality factor maintainability into two quality sub-factors or criteria which are **Ease of Migration** and **Ease of Replacement**. The quality measure to measure Ease of Migration is **Migration Ease Level** and the quality measure to measure Ease of Replacement is **Replacement Ease Level** respectively.
Both quality measures will be level measures. On receiving a new version of the COTS component, the end user might also want to know how stable the component is after the modifications done to the previous version of the COTS component. We include a quality criterion called **Stability** to express this behavior characteristic. To measure stability we have proposed and defined a **COTS Component Stability Metric** which will be explained in chapter 5. The schematic for the quality factor maintainability is shown in Figure 4.4.

### 4.8.2.2 Quality factor 2: testability

We have introduced testability as a high level quality factor in the proposed quality COTS quality model with its own set of quality criteria and measures. The need for testability as a high level factor has already been discussed in section 2.5.2 in chapter 2. Testability is a quality factor which can be predicted as soon as the system is specified; to indicate the ease of testing the system (Nguyen et al 2002). Testability oriented specification provides helpful information about the testing effort needed for reaching quality requirements (Traon and Robach 1997). Since COTS components are like black boxes, a highly testable component will be more reliable (Voas et al 1995).

One of the necessary requirements for a COTS component to be testable is the presence of proper test documentation (Gao 1999). Therefore
we have chosen **Test Documentation** has one quality criteria. Test documentation can either be documentation of the test suit or the documentation proof of previous tests that have been conducted on the COTS component. So, we have two measures under test documentation, one being **Test Suit Documentation** and the other being **Proofs of Previous Tests** respectively.

Another factor that is directly related to testability is controllability because controllability directly affects testability (Gao et al 2002). Testers and customers expect components to provide a set of control functions in software components so that they can use them to check diverse component behaviors according to their needs (Gao 1999). Freedman (1991) has also introduced the testability notion by defining controllability. **Controllability** is another quality sub-factor/criterion under testability. We have proposed three measures which are **Component Execution Control**, **Component Environment Control** and **Component Function Feature Control** to measure controllability. All three measures proposed are Presence metrics. Figure 4.5 shows the schematics for the quality factor testability.

![Figure 4.5 Schematic for the Quality Factor Testability](image-url)
Testability is also determined by Traceability (Gao 1999) which is also included as a quality criterion. There are six types of traces namely, operation, performance, error, state, GUI Events and communication traces (Gao 1999). Of all the six, performance and error trace can easily be measured by end users and are sufficient to state whether the component is of a high quality or not (Gao et al 2002). So, the quality criterion traceability has the quality measures Performance trace and Error trace and both are Presence metrics.

4.8.2.3 Quality factor 3: functionality

Functionality is the quality attribute that relates to the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs (ISO/IEC 2001). We include functionality as a high level quality factor in the model on the basis of Alvaro’s surmise (2005) that functionality can also be expressed as the ability of the component to provide the required services when used under specified conditions. Alvaro et al (2005) have included self contained as a quality criterion which is justified since self contained is a very important component property (Szyperski 2007) and can be applied to the COTS domain also. Szyperski (2007) says that a self contained component should have well defined interfaces and should not be dependent on other components to carry out it’s functionalities. A well defined interface means that the interface of the component should have well defined preconditions and post conditions (Szyperski 2007). We have also included Self Contained as a criterion and have proposed the quality measure Presence of Preconditions and Post conditions which is a Presence metric to check if preconditions and post conditions have been defined. To check for dependence on other components we use the quality measure Modularity that has been proposed by Alvaro et al (2005) which is a ratio metric.
Another quality criterion called **Compliance** proposed by Alvaro et al (2005) has also been used in this model. Compliance has two quality measures, **Presence of Standardization** and **Presence of Certification** (Alvaro et al 2005) which are presence metrics.

Finally, Alvaro et al (2005) have suggested several criteria and measures such as precision, correctness and accuracy. Our aim is to include only the minimum necessary attributes required to evaluate COTS quality in our model. Therefore we have included quality criterion **Accuracy** and quality measure **Correctness** in the proposed model. Figure 4.6 shows the schematic for the quality factor Functionality.

![Figure 4.6 Schematic for Functionality](image)

### 4.8.2.4 Quality factor 4: efficiency

Efficiency is the set of attributes that relate to the relationship between the level of performance of the software and the amount of resources
used under stated conditions (ISO/IEC 2001). The ISO quality model itself subdivides efficiency into the following quality criteria:

1. **Time behavior**: Attributes of software that relate to response and processing times and on throughput rates in performing its function.

2. **Resource behavior**: Attributes of software that relate to the amount of resources used and the duration of such use in performing its function.

The same has been followed in all the proposed COTS quality models. The quality measures suggested are disk capacity and response time (Bertoa and Valecillo 2002). Since there is not much change required to apply these characteristics to the COTS domain, we choose the same quality criteria and measures for the Q’Facto 10 model also for the high level quality factor efficiency. Figure 4.7 is the schematic for the efficiency quality factor.

![Figure 4.7 Schematic for Efficiency](image)

### 4.8.2.5 Quality factor 5: reliability

Reliability is the set of attributes that relate to the capability of software to maintain its level of performance under stated conditions for a stated period of time (ISO/IEC 2001). In the ISO quality model, reliability has three criteria namely, maturity, recoverability and fault tolerance. Maturity is the number of faults in the software and can be determined only by the developer and not the end user.
Figure 4.8 Schematic for Quality Factor Reliability

We therefore include only Recoverability and Fault tolerance as quality criteria in our model. We use the measures Presence of Exception Handling and Persistence to measure recoverability and Presence of Fault Tolerant Mechanism to measure Fault Tolerance. All three are presence measures. Figure 4.8 is the schematic for the reliability quality factor.

4.8.2.6 Quality factor 6: usability

Usability is the set of attributes that relate to the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users (ISO/IEC 2001). Usability has been included in the proposed Q'Facto quality model because usability is an important customer or end user oriented quality factor or attribute (Patrick Berander et al 2005). The ISO model has understandability and learnability has quality criteria for usability.

Figure 4.9 Schematic for Usability
Understandability relates to the users' effort for recognizing the logical concept and its applicability while Learnability relates to the users' effort for learning its application (for example, operation control, input, and output). Our aim is to create a comprehensive model. To evaluate usability, it is necessary that the component has a good Help system and other facilities that aid the end user to study the component better (Dukic and Boergh 2003). We have decided to include just the quality criterion **Learnability** with the quality measures **Help System, Training** and **Demonstration Coverage**. Figure 4.9 is the schematic for the usability quality factor.

### 4.8.2.7 Quality factor security

We have added security has a high level quality factor in the proposed model. A COTS component should be able to prevent unauthorized access whether accidental or deliberate to its programs and data. We have included single sub-factor/criterion called **Access Control** that has been modified to suit the COTS domain. Access Control indicates whether the COTS component is able to control access to its interfaces. We include a quality measure **Access Controllability** to measure access control which is a presence metric. Figure 4.10 is the schematic for the security quality factor.

![Figure 4.10 Schematic for Security](image)

### 4.8.2.8 Quality factor 8: portability

Portability refers to the ability of software to be transferred from one place to another. Only the ISO sub-factors **Installability** and adaptability can be included and adapted to the COTS domain and therefore are included in the model. COTS components can be installed as well as deployed. Installability and deployability are two totally different things. A COTS
component can be installed once and deployed many times (Szyperski 2007). So, we have included **Deployability** as a quality sub-factor in the Q'Facto 10 quality model. **Adaptability** has been included because COTS components come in containers and can be transferred from one container to another. The quality measures are **Installability Documentation** and **Installability Complexity** for quality criteria Installability, **Deployability Documentation** and **Deployability Complexity** for quality criteria Deployability and **Mobility** for quality measure Adaptability. Figure 4.11 is the schematic for the quality factor Portability.

![Figure 4.11 Schematic for Portability](image)

**4.8.2.9 Quality factor 9: interoperability**

Cots Components usually have to interact with one another in COTS based system. Hence, the inclusion of interoperability as a high level quality factor in our model. Interoperability has the sub-factor **Compatibility** which can be measured by **Data Compatibility** or **Version Compatibility**. Figure 4.12 is the schematic for the quality factor Interoperability.
4.8.2.10 Quality factor 10: reusability

According to Hopkins John (2000), reusability is important in the development of a component-based system. Hopkins John (2000) 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 (Hopkins Jon 2000). There are expectations that a high quality component has to be reusable. Hence, it is becomes necessary to include reusability as a high level quality factor in our quality model.

Reusable code has to be generic in nature (Caroll and Ellis 1995). So Generality has been included as a quality criterion which is measured by the quality measure Presence of Domain Abstraction. Hardware and software independence will also determine the reusability of a COTS component (William and Carol 1996). Therefore Hardware/Software Independence is included as a quality criterion which is measured by the quality measures Presence of Hardware independence and Presence of Software Independence respectively. Finally, it should be easy to locate a reusable component from a repository. So our third quality criterion and measure are Locatability and Accessibility respectively. Figure 4.13 is the schematic for the quality factor Reusability.
Figure 4.13 Schematic for Reusability

4.9 QUALITY CALCULATOR

The model can be used to calculate the overall quality of a COTS component.

4.9.1 Assumptions

The following points are to be noted:

i. 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.

ii. Weights are assigned to each quality factor and quality measure and the component user may choose these weights.

iii. 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 and we classify it as a low quality COTS component.

### 4.9.2 Calculation

The steps to be followed for measuring the quality of the component are as follows:

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

   \[
   QF_i = \frac{\sum_j \sum_k (QM_{i,j,k} \times W_{i,j,k})}{\sum W_{i,j,k}}
   \]  \hspace{1cm} (4.1)

   where

   \( QM_{i, j, k} \) = value corresponding to the \( k^{th} \) quality measure of the \( j^{th} \) quality criterion belonging to the \( i^{th} \) quality factor E.g. \( QM_{1, 1, 1} \) is the quality measure component stability

   \( W_{i,j,k} \) = value corresponding to the weight assigned to \( k^{th} \) quality measure of the \( j^{th} \) quality criteria belonging to the \( i^{th} \) quality factor

   \( N_{i,QM} \) = the total number of quality measures for the Quality Factor \( QF_i \)

2. 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.

3. Steps 2, 3, 4 should be repeated for each of the remaining quality factors of the model.
4. If the component passes in all the factors then the overall quality of the component can be calculated using the formula

\[
\text{Overall Quality} = \frac{\sum^{10}_{i=1} QF_i \times W_i}{\sum W_i}
\]

(4.2)

where

\( W_i \) = value corresponding to the weight assigned to the quality factor \( QF_i \)

\( QF_i \) = value of Quality Factor i

5. 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.

The flow chart in Figure 4.14 gives a description of the calculation process.

![Quality Calculation Flow Chart](image)

**Figure 4.14** Quality Calculation Flow Chart
4.10 ILLUSTRATIVE STUDY

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 tabulated below in Table 4.1. The quality measure values are average of responses from the five members and a uniform weight of 1 is considered for all the quality factors.

Table 4.1 Component Quality Evaluation Study Results

<table>
<thead>
<tr>
<th>Quality Measure</th>
<th>JGantt</th>
<th>GanttBiz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component Stability (Maintainability)</td>
<td>0.6</td>
<td>0.5</td>
</tr>
<tr>
<td>Migration Ease Level (Maintainability)</td>
<td>0.6</td>
<td>1.0</td>
</tr>
<tr>
<td>Replacement Ease Level (Maintainability)</td>
<td>0.8</td>
<td>1.0</td>
</tr>
<tr>
<td><strong>Quality Factor Maintainability</strong></td>
<td><strong>0.67</strong></td>
<td><strong>0.83</strong></td>
</tr>
<tr>
<td>Test Suit Documentation (Testability)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Proofs of Previous Tests (Testability)</td>
<td>1.0</td>
<td>0</td>
</tr>
<tr>
<td>Component Execution Control (Testability)</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Component Environment Control (Testability)</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Component Function Feature Control (Testability)</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Performance Trace (Testability)</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Error Trace (Testability)</td>
<td>0</td>
<td>1.0</td>
</tr>
<tr>
<td><strong>Quality Factor Testability</strong></td>
<td><strong>0.71</strong></td>
<td><strong>0.71</strong></td>
</tr>
<tr>
<td>Presence of Preconditions and Post conditions (Functionality)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Modularity (Functionality)</td>
<td>1.0</td>
<td>0.888</td>
</tr>
<tr>
<td>Presence of Standardization (Functionality)</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Presence of Certification (Functionality)</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Computational accuracy (Functionality)</td>
<td>0.92</td>
<td>0.98</td>
</tr>
<tr>
<td><strong>Quality Factor Functionality</strong></td>
<td><strong>0.78</strong></td>
<td><strong>0.77</strong></td>
</tr>
<tr>
<td>Disk Capacity (Efficiency)</td>
<td>0.8</td>
<td>1.0</td>
</tr>
<tr>
<td>Response Time (Efficiency)</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td><strong>Quality Factor Efficiency</strong></td>
<td><strong>0.9</strong></td>
<td><strong>1.0</strong></td>
</tr>
<tr>
<td>Persistence (Reliability)</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Presence of Fault Tolerant Mechanism (Reliability)</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 4.1 (Continued)

<table>
<thead>
<tr>
<th>Quality Factor</th>
<th>JGantt</th>
<th>GanttBiz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Help System (Usability)</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Training (Usability)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Demonstration Coverage (Usability)</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Usability Factor</td>
<td>0.66</td>
<td>0.66</td>
</tr>
<tr>
<td>Access Controllability (Security)</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Installability Documentation (Portability)</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Installability Complexity (Portability)</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Deployability Documentation (Portability)</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Deployability Complexity (Portability)</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Mobility (Portability)</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Portability Factor</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Data Compatibility (Interoperability)</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Version Compatibility (Interoperability)</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Interoperability Factor</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Presence of Domain Abstraction (Reusability)</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>History of reuse (Reusability)</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Presence of Hardware Independence (Reusability)</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Presence of Software Independence (Reusability)</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Accessibility (Reusability)</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Reusability Factor</td>
<td>0.6</td>
<td>0.4</td>
</tr>
</tbody>
</table>

The measures have been calculated using the metrics that will be discussed in the next chapter. For example, let us consider the calculation of the Quality Factor testability. There was no test suit documentation or error log available for the JGantt software. So it has a value for both the metrics. On the other hand Proofs of previous tests, component Function Feature Control and all the other measures are available and each of the metric has a value of 1. So the quality measures calculated will be
1) Test Suit Documentation = value * weight =0*1 =0
2) Proofs of Previous Tests = value *weight =1*1 =1
3) Component Execution Control = value *weight =1*1 =1
4) Component Function Feature Control = value *weight =1*1 =1
5) Component Environment Control = value *weight =1*1 =1
6) Performance Trace = value *weight =1*1 =1
7) Error Trace = value * weight =0*1 =0

Considering a uniform weight of 1 for all the measures, the value of the testability factor for the JGantt software is $(1+1+1+1+1)/7 =0.71$. Similarly, the value for the testability factor for the Gantt project is 0.71. After calculating the individual quality factors and considering a uniform weight of 1 for all the quality factors for all three software components and using the equation (4.2),

Overall quality factor for the JGantt component = 0.832

Overall quality factor for the GanttBiz component = 0.825

In this example, we have given a uniform weight of 1 to all the quality factors and measures. 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. We have also chosen a threshold value of 0.5 for each quality factor. Since, both the components have not obtained a value lesser than 0.5 for any of the quality factor, we can say that they both are quality components.
4.11 CONCLUSION

A comprehensive quality model that could be used by user groups for evaluating the quality of COTS components has been proposed in this chapter. 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. The model was submitted to journals for critical reviews and as per suggestions from reviewers of our publications changes have been made to the proposed Q'Facto model. These changes and the resultant Q'Facto 12 model are discussed in the next chapter.