CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

The development of complex, real time software systems with very high availability and reliability requirements entail enormous costs (Pour 1999). This has made organizations to consider implementing such systems using Commercial-Off- the- shelf (COTS) components with the expectation that COTS components can significantly lower development costs and shorten development cycles (Weyuker 1998). Organizations also expect that using COTS components will ultimately lead to software systems that require less time to specify, design, test and maintain and yet still maintain high reliability requirements. Many organizations see the arrival of platforms like COM, DCOM and Net Beans IDE which can be used to implement software components as the silver bullet that they have been waiting for decades. They see components as building blocks which can be easily incorporated into a software system to provide a specified functionality.

For example, let organization XYZ, be assigned the task of creating the software for running a shopping mall. Assuming that they have a deadline of a year and all the modules are within the delivery deadlines except the purchase and accounts module which has not been started yet. Organization XYZ can either form a new team to work on the module or can purchase the COTS component for the module if it is available and integrate it into their existing system. This will save the organization development time and
development cost. In spite of the obvious benefits that COTS components offer us there are still a number of questions as to the effectiveness of the Component Based Software Development (CBSD) strategy. Some of the most important questions are

1. Will the COTS component fit into the system without damaging or harming the already existing system?

2. Has the COTS component been effectively tested by the component vendor before its appearance on the component market?

3. Is the COTS component reliable?

4. If there are problems later with the COTS component will it be easy to correct the problems and maintain the component with ease?

5. Does the COTS component satisfy the requirement of the component purchaser or end user?

As can be seen, these questions revolve around the quality of the purchased COTS components. Therefore, the quality of COTS components is a very important issue because the success of the component based approach depends on choosing high quality components. Component technology is a technology that develops a system through acquisition and assembly. The component based system is largely dependent on the quality of each component that comprises the system. If low quality components are used, they might cause damage to the existing system leading to an avalanche affect. For example, if the purchase component in the example above fails, the team’s entire work of one year becomes wasted. Therefore to realize the
potential benefits of COTS components it becomes very important to evaluate the quality of COTS components before purchasing them.

Many research institutions have been studying methodologies for component development and standardization. However, from the consumer’s viewpoint there is little achievement in the research of quality evaluation of COTS components (Gao and Wu 2003). The work presented in this thesis is an effort to provide a minimum basic quality model that can be used for the evaluation of COTS components by the users of the COTS components.

1.2 RESEARCH OBJECTIVE

The objective of this research is to develop a comprehensive quality model suitable for use by end users that can assess the quality of Commercial Off-The-Shelf components. Effort has to be focused in the following directions to achieve the above objective:

1. The first step should be to demarcate clearly COTS components from the rest of software components and find exact specifications for COTS components.

2. The second step is to develop suitable quality models for the COTS components so defined.

3. The third step will be to assess the validity of the above model using suitable statistical tests.

1.3 RESEARCH CONTRIBUTION

As a first step, a literature survey of the existing component quality models was carried out. It is observed that with the continuing rise in the demand for software component based products, the terms - component, in-
house component, COTS and reusable components have become overloaded and conflicting usage of the terms are widely prevalent. Therefore, before attempting to build a quality model for quality evaluation, it becomes necessary to distinguish between different kinds of components and formally define a COTS component. Accordingly, a nomenclature called the TRUSTAD nomenclature has been proposed that distinguishes In-House and COTS components and a formal definition and specification is provided for COTS components. A quality model called Q'Facto 10 was first constructed using the ISO 9126 (ISO/IEC 2001) quality standard as a foundation and the above definition and nomenclature as a defining basis for COTS components. Based on peer review and critical suggestions provided by journal reviewers, the Q'Facto10 model has been upgraded to the Q'Facto 12 quality model using the ISO 25000 [SQuARe] quality standard (ISO/IEC 2005). The final step was to validate the model using inputs from software professionals and carrying out statistical analysis of the input using the Student’s t Test. The statistical tests establish the basic soundness of the model proposed.

1.4 SOFTWARE COMPONENT DEFINITION

Szyperski et al (2007) define a component as, "A software component is a unit of composition with contractually specified interfaces and explicit context dependencies. A software component can be deployed independently and is subject to composition by third parties." Not all software components can be subject to composition by third party vendors. Hence, the above definition though technically correct, is not complete. Michael Sparling (2002) defines a component, "as a language neutral, independently implemented package of software services, delivered in an encapsulated and replaceable container accessed via one or more published interfaces. While a component may have the ability to modify a database, it cannot be expected to
maintain state information. A component is not platform constrained nor is it application bound." Only recently available technologies like JavaBeans and EJB deploy components in containers and it is not a given rule that all components exist in containers. So, we disagree with this definition also. Let us try to understand what a software component is before we present our definition for a component.

Let us consider an analogy first. When we purchase a computer, we generally buy different parts from different vendors and assemble them together. We may get the hard disk from Seagate, monitor from Samsung, the keyboard from HCL and we can top it all with a P4 processor from INTEL. The general idea is that we get the components suited to our specifications and by assembling them all together, the final product is cost effective as compared to a wholly branded model that may not snug fit to our design specifications. Even in the automobile industry, the different parts of a car are manufactured by different vendor factories and then assembled together to get a final product thus cutting down on production costs.

On the other hand, let us consider a software program for an Airline Reservation System with separate modules for ticket reservation, ticket cancellation and ticket availability. A decade ago, it would have been impossible to get each module developed by a different vendor and then assemble them together to get a fully integrated product (Heineman and Councill 2001). This is because software can be written in different languages and different platforms and it was not possible for software written in one platform to be integrated with software written in another platform (Goulao and Brito 2002). This brought about the component revolution.

Component technology allows software modules written in different technologies to be integrated with one another with the help of
middleware technologies like COM, DCOM, CORBA, JavaBeans and EJB. These middleware technologies are simply a set of specifications or rules in the form of functions, which when incorporated into the code allows the software to be integrated with software developed using other platforms/languages (Sparling 2000).

To explain further, if the ticket reservation module in an airline reservation system had been written using VB following the COM/DCOM specifications we could have integrated it with any other airline reservation application program written in any other language/platform that needed a ticket reservation module. This saves development time for the programmers and may also provide cost benefits for the organization (Goulao and Brito 2002).

Therefore, there are a few things which should be taken into consideration before framing a definition for a component. The first important point to remember is that once a piece of software has been written using such a middleware specification it can be integrated only if it has already been executed and tested. In this process, we identify two distinct groups of persons, component developers and who develop the components and component users who put the components into use.

The second point to remember is that the component user cannot access the software component directly but only through a set of well published interfaces for security reasons (Carney and Long 2006). Developers would not like every person to have access to their source code for obvious reasons. A component’s interfaces are always independent of the implementations (Crnkovic et al 2002).
The third point to remember is that components created using middleware technologies like JavaBeans and EJB are encapsulated inside a container.

Consolidating the above three points, we present our definition of a component by definition 1.1.

**Definition 1.1** “A component is any piece of independently executable binary code written to a specification, which can only be accessed via a set of well published interfaces and which can be integrated into any kind of software application irrespective of language/platform. A component always offers a set of services via its interfaces and may be encapsulated inside a container depending on the kind of middleware technology used to develop the component.”

Now that we have formally defined components, let us have a look at the different types of components.

**1.5 SOFTWARE COMPONENT TYPES**

Software components can be broadly classified into two categories: In-House components that are developed inside the organization itself and COTS (commercial off the shelf) components that are purchased from third party vendors. Figure 1.1 shows the two types of software components.

![Software Component Types](image-url)
Our research primarily focuses on COTS components and the study and design of a quality model that can be used as a standard for evaluating COTS components. To fully appreciate the importance for the need of customer quality evaluation of COTS components, a few case studies are been highlighted in the next section.

1.6 CASE STUDIES

Our first example is the often referenced Ariane disaster in component literature (Jezequel and Meyer 1997). In June 1996, during the maiden voyage of the Ariane 5 launch vehicle, the launcher veered of course and exploded less than one minute after takeoff. The inquiry board convened by ESA’s director general and the CNES’s chairman determined that the explosion resulted from failure of a component that had not been adequately evaluated with the Ariane 5 software because it was reused from the Ariane 4 launcher. Although the component had worked perfectly in Ariane 4, it had led to problems in Ariane 5. The developers had assumed that the component would work perfectly well since there was no significant difference between the two systems.

Our second example involves an award winning design and management tool vendor (and component consumer) who implemented an ORBIX daemon for client to database communications in it’s application’s architecture (Weyuker 1998). The daemon worked flawlessly in the application’s first version. The producer then demanded that the daemon’s consumers upgrade to a more efficient version. Trust had developed between the producer and consumer and the vendor implemented the new daemon. Customers were made aware of the upgrade and all awaited the increased performance. The daemon was exceptionally defective and the applications
mean time between failures at some customers’ sites decreased from months to hours.

Finally, when Adobe released photoshop3.0, it forgot to remove a time bomb that automatically shut the program down. The time bomb was an overlooked remnant of the beta test cycle. All the above incidents could have been avoided if there had been some kind of quality evaluation from the customer’s side before distributing the component. If such a quality evaluation standard existed, then, developers also would design the component having the quality requirements of the customer in mind (Voas 1998b).

The next section presents existing COTS component standards.

1.7 EXISTING COTS COMPONENT STANDARDS

The first standard was UL1998 framed by the Underwriters Laboratory. Underwriters Laboratories’ (UL) mission is to address consumers’ needs for safety and quality through independent assessment of conformance to standards. UL, a nonprofit organization is ordinarily recognized for the testing of and standards for products whose failure can result in fire, electric shock or personal injury. To address the emergence of a component based software economy, UL undertook a five year component testing study that resulted in an initial protocol and adjunct service for reusable software components. This service which is centrally managed out of the UL Research Triangle Park Laboratory is based on the Air Force Rome Laboratory Framework for certification of reusable software components (Gao and Wu 2003).
The UL standard for Safety related software UL1998 was first published in 1994. The UL 1998 second edition was re titled the Standard for Software in Programmable Components to reflect more accurately it’s scope and was released in May1998. The second edition which achieved ANSI status in February of 1999, applies to non-networked embedded software in programmable(microelectronics)components(Gao and Wu 2003).

The British Standards Institution (BSI) has developed BS 7925-2, which is a standard for testing software components where a software component is defined to be “a minimal program for which a separate specification is available.” However, we note again that this definition of software component is less stringent than ours. We also note that the test techniques specified in BS7925-2 pertain mostly to white-box testing, and therefore, the standard is useful as a guide for the testing process for software components during their development and certification (Gao and Wu 2003).

The above two standards are standards related to the component development and testing processes only. Both the standards help as guides during the component testing and development phases. They do not help to evaluate component quality after development is over and the component is ready for purchase by a buyer. In addition to these standards, the Butler Group has introduced the concept of component based maturity development levels (Voas 1998a).

The focus was again on the development process and not only that this capability maturity framework was conceived focusing on large organizations with complex development projects. It is a question whether small IT organizations will adopt the Butler’s Group’s attempts to construct a propriety model to enhance component quality.
In 1997, a committee was gathered to work on the development of a proposal for an IEEE standard on software components product oriented quality from the customer’s point of view. The initiative was later suspended, since the committee came to a consensus, that they were still far from getting to a point where the document will be a strong candidate for a standard (Goulao and Brito 2002). The failure was due to number of reasons which will be explained in the next section.

Therefore, as of today, no standard exists for COTS component quality which can be used as a guideline for evaluating the quality of a COTS product from the consumer’s point of view. The difficulties we might face in bringing out such a standard are explained in the next section.

1.8 PROBLEMS FACED IN COTS QUALITY EVALUATION

The first issue is the lack of information provided by COTS vendors (Alvaro et al 2005). A copyrighted COTS component is usually sold only as an executable file and the source code is not available to the component consumer. Some documentation will be made available to the user but it is usually not sufficient to test for the quality of the component. It will not be possible to use this information to measure some quality attributes like extensibility etc.

The second issue is the heterogeneous nature of components (Lau and Wang 2007). A component can be used in multiple platforms and applications and the behavior of the component will change from application to application. Even in the Ariane case study given above the component is of high quality when used with Ariane 4, but fails as a high quality component when used with Ariane 5. So, when the same component shows such a difference in behavior between two versions of the same system, one can
imagine how varied the behavior would be for different applications. As such, it becomes too difficult to assess the quality.

The third issue which is closely related to the above problem is the difficulty in gaining a consensus between different users as to the quality of the component depending on the specific use of the component (Gao and Wu 2003). For example, a component used to create charts can be used in different ways. Assuming, the component draws charts perfectly well, but does not have a well developed printer and document import /export interface. Consider two groups of users A and B of the same component. User group A might need the component only to generate charts while on the other hand, user group B might need the component to generate, print and also to import and export chart diagrams. Therefore, according to user group A, the component will be a high quality component with respect to the functionality quality attribute. However, user group B will declare the component to be a low quality component because of it’s failure to satisfy the functionality of printing and import/export, although it draws charts perfectly well.

The fourth issue is that requirements vary from organization to organization and it is utopia to create a component that will satisfy all the requirements (Goulao and Brito 2002). A component can pass the quality requirements of one organization and fail in the quality requirements set by another organization.

The fifth issue is the question of who gets to evaluate the quality of the COTS product (Alvaro et al 2005). Should the COTS product be evaluated by a third party organization or by the component user itself? If evaluated by a third party organization, a number of liability issues will be raised. Finally, since a standard may take several years to develop and software components are a relatively recent phenomenon (Gao and Wu 2003).
We have had a look at all the issues and difficulties in setting standards for COTS quality evaluation. We have also seen the problems and disasters that can take place when low quality COTS components are used. From these, we come to the conclusion that in order, to take the best advantage of the component based development strategy and to move forward in the positive direction, it become imperative and unavoidable that some kind of standards for quality evaluation of COTS components should be framed. This can be done only if necessary steps are taken now itself, so that at least a foundation for such a standard can be laid now and the required super structure can be built upon later in the years to come.

The next section talks about how the discussed issues can be overcome.

1.9 OVERCOMING THE ISSUES

The first step that has to be taken towards a standard for COTS quality using a product oriented approach from the consumer’s point of view is to define a standard quality model that can be used by all organizations to evaluate the quality level of a COTS product. To overcome the issues that the vendor does not provide enough information and the heterogeneous nature of the platforms, a set of quality attributes should be chosen in such a way that the chosen quality attributes should apply uniformly to all COTS components and the quality attributes can be measured with the usual information given by the component vendor. In this regard, the ISO specifications (ISO/IEC 2004) are useful as they provide the framework for the basic quality attributes like functionality, usability, maintainability, security, portability, reliability and efficiency that are common for all COTS products. Of course, changes should be made to these attributes so that they can be applied to the COTS domain. Additional quality attributes can also be added.
We also need a model that can be used by both component users as well as third party organizations. Therefore the model should only have product oriented quality attributes and quality measures.

Since, different users have different perspectives of the same components, their measurement and importance of each attribute will be different for each of them. A weighted scoring method (Gillies 2003) can be used which allows the user to give a weight to each quality attribute according to how important they consider the attribute is to the quality of the component. Each attribute is evaluated to produce a score between 0 and 1. Each score is weighted before summation and the resultant score reflects the relative importance of the different factors. The component fails to qualify as a high quality component if the total of any attribute is lesser than a given threshold.

The focus of our research is to create a quality model that could be used by consumers to assess COTS component quality before integrating the COTS component with their software systems. A literature survey of existing COTS quality model has been presented in chapter 2. All the proposed quality models are based on the ISO 9126 quality model.

We shall therefore have a look at the different software quality models that have been proposed and understand why the ISO 9126 model has been chosen as the framework for all the proposed quality models. First, we first have to get a clear idea of the software perspective before studying the models and this has been described in the next section.

1.10 SOFTWARE QUALITY PERSPECTIVE

Kitchenham and Pfeeger (1996), by referring to the teachings of David Garvin, report on the five different perspectives of quality:
1. The transcendental perspective deals with the metaphysical aspect of quality. In this view of quality is “something toward which we strive as an ideal, but may never implement completely.” (Kitchenham and Pfleeger 1996)

2. The user perspective is concerned with the appropriateness of the product for a given context of use. Kitchenham and Pfleeger (1996) further note that “whereas the transcendental view is ethereal, the user view is more concrete, grounded in the product characteristics that meet user’s needs”.

3. The manufacturing perspective represents quality as conformance to requirements. This aspect of quality is stressed by standards such as ISO 9001, which defines quality as “[the] degree to which a set of inherent characteristics fulfils requirements” (ISO/IEC, 1999).

4. The product perspective implies that quality can be appreciated by measuring the inherent characteristics of the product. Such an approach often leads to a bottom-up approach to software quality: by measuring some attributes of the different components composing a software product, a conclusion can be drawn as to the quality of the end product.

5. The final perspective of quality is value-based. This perspective recognizes that the different perspectives of quality may have a different importance, or value, to various stakeholders (Georgiadou 2003).

We now have a look at existing software quality models.
1.11 **SOFTWARE QUALITY MODELS**

This section presents a study of the different quality models proposed for software and discusses why the ISO 9126 is the best model proposed so far.

1.11.1 **Mccall’s Quality Model (1977)**

One of the more renowned predecessors of today’s quality models is the quality model presented by Jim McCall et al (1977) (also known as the General Electrics Model of 1977). This model, as well as other contemporary models, originate from the US military and is primarily aimed towards system developers and the system development process. In his quality model McCall attempts to bridge the gap between users and developers by focusing on a number of software quality factors that reflect both the users’ views and the developers’ priorities.

The McCall quality model in Figure 1.2 uses three major perspectives for defining and identifying the quality of a software product: **product revision** (the ability to undergo changes), **product transition** (the adaptability to new environments) and **product operations** (operation characteristics).

![Figure 1.2 Overview of McCall’s Quality Model](image-url)
Product revision includes: (1) maintainability (the effort required to locate and fix a fault in the program within its operating environment), (2) flexibility (the ease of making changes required by changes in the operating environment) and (3) testability (the ease of testing the program, to ensure that it is error-free and meets its specification). Product transition is all about: (1) portability (the effort required to transfer a program from one environment to another), (2) reusability (the ease of reusing software in a different context) and (3) interoperability (the effort required to couple the system to another system). Product operations depend on: (1) correctness (the extent to which a program fulfills its specification), (2) reliability (the system's ability not to fail), (3) efficiency (further categorized into execution efficiency and storage efficiency and generally meaning the use of resources, e.g. processor time, storage), (4) integrity (the protection of the program from unauthorized access) and (5) usability (the ease with which the software can be used).

The model furthermore details the three types of quality characteristics (major perspectives) in a hierarchy of 11 factors that describe the external view of the software, 23 criteria that describe the internal view of the software and metrics that provide a scale and method for measurement.

![Figure 1.3 Quality Factors and Criteria of Mccall’s Quality Model](image-url)
Figure 1.3 (Continued)
The idea behind McCall’s Quality Model is that the quality factors synthesized should provide a complete software quality picture. The actual quality metric is achieved by answering yes and no questions that then are put in relation to each other. Figure 1.3 outlines the quality factor and criteria of McCall’s quality model.

1.11.2  **Boehm’s Quality Model (1978)**

The second of the basic and founding predecessors of today’s quality models is the quality model presented by Barry W. Boehm (1978). Boehm addresses the contemporary shortcomings of models that automatically and quantitatively evaluate the quality of software. In essence his model attempts to qualitatively define software quality by a given set of attributes and metrics.

As seen in Figure 1.4 the high-level characteristics represent basic high-level requirements of actual use to which evaluation of software quality could be put and addresses three main questions that a buyer of software has:

1. **As-is utility:** How well (easily, reliably, efficiently) can I use it as-is?
2. **Maintainability:** How easy is it to understand, modify and retest?
3. **Portability:** Can I still use it if I change my environment?
1.11.3 Dromeys’s Quality Model

A comparatively recent model similar to the McCall’s and Boehm’s quality model is the quality model presented by Geoff Dromeys (1995). Dromeys proposes a product based quality model that recognizes that quality evaluation differs for each product and that a more dynamic idea for modeling the process is needed to be wide enough to apply for different systems. Dromeys is focusing on the relationship between the quality attributes and the
sub-attributes, as well as attempting to connect software product properties with software quality attributes.

In Figure 1.5, we present the structure of Dromey’s quality model which encompasses the following:

1) Product properties that influence quality
2) High level quality attributes
3) Means of linking the product properties with the quality attributes.

Dromey's Quality Model is further structured around a 5 step process:

1) Choosing a set of high level quality attributes necessary for the evaluation.
2) Listing components/modules in the system.
3) Identifying quality carrying properties for the components/modules (qualities of the component that have the most impact on the product properties from the list above).
4) Determining how each property affects the quality attributes.
5) Evaluating the model and identifying weaknesses.

![Figure 1.5 The Structure of Dromey’s Quality Model](image-url)
1.11.4 The ISO Quality Model

ISO 9001 is an international quality management system standard applicable to organizations within all type of businesses. Besides the famous ISO 9000, ISO has also released the ISO 9126: Software Product Evaluation: Quality Characteristics and Guidelines for their Use-standard (ISO/IEC 2001). This standard was based on the McCall and Boehm models. Besides being structured in basically the same manner as these models as seen in figure 1.5, ISO 9126 also includes functionality as a parameter, as well as identifying both internal and external quality characteristics of software products.

![ISO Quality Model Diagram](image)

**Figure 1.6 The ISO Quality Model**

The ISO 9126 proposes a standard which specifies six areas of importance, i.e. quality factors for software evaluation.

The quality model defined in ISO/IEC 9126-1 recognizes three aspects of software quality and defines them as follows:
1) **Quality in use**

Quality in use is the user’s view of the quality of the software product when it is used in a specific environment and a specific context of use. It measures the extent to which users can achieve their goals in a particular environment, rather than measuring the properties of the software itself (ISO/IEC 2001).

2) **External quality**

External quality is the totality of characteristics of the software product from an external view. It is the quality when the software is executed, which is typically measured and evaluated while testing in a simulated environment with simulated data using external metrics. During testing, most faults should be discovered and eliminated. However, some faults may still remain after testing. As it is difficult to correct the software architecture or other fundamental design aspects of the software, the fundamental design remains unchanged throughout the testing (ISO/IEC 2001).

3) **Internal quality**

Internal quality is the totality of characteristics of the software product from an internal view. Internal quality is measured and evaluated against the internal quality requirements. Details of software product quality can be improved during code implementation, reviewing and testing, but the fundamental nature of the software product quality represented by the internal quality remains unchanged unless redesigned (ISO/IEC 2001).

The internal and external quality model is inspired from McCall and Boehm’s work. It is a three-layer model composed of quality characteristics, quality sub-characteristics and quality measures.
Though several quality models have been suggested, it can be seen that many of these are not able to fully capture the intuitive feeling of quality of software the user gets. The missing of this transcendental perspective has been pointed out by Alexis Cote et al (2007).

According to Alexis cote et al (2007), the ISO 9126 model seems to recognize all the perspectives of quality as important contributors to the overall assessment of quality. It takes an incremental approach to software quality that begins with quality in use, something that is easy to grasp for non-technical stakeholders and ends with internal quality, something more technically inclined stakeholders will feel more comfortable with. Furthermore, there is a comprehensive set of suggested measures that allow for the assessment of software quality. ISO/IEC 9126 is thus the only model that fulfils all the stated requirements for a model to be useful as a foundation to Software Quality Engineering (Alexis Cote et al 2007).

1.12 Decision Making Methods

There are different methodologies that can be used with quality models to evaluate the quality of software products. These methods are grouped as Multi-Criteria Decision Analysis (MCDA) methods (Morisio and Tsoukios 1997). They are also alternatively called as Multi-Criteria Decision Making (MCDM) methods. These methods have been used with an amount of success in decision-making environments in many disciplines and are useful in helping decision makers when they encounter conflicting view points. Some of the most popular MCDM methods are –Weighted Sum Model (Triantaphyllou. E. 2000), Weighted Product Model (Triantaphyllou 2000), the group of ELECTRE (Et Choix Traduisent la Realite-Elimination and Choice Expressing Reality) models (Roy and Bernard 1968), Preference Ranking Organization Method for Enrichment Evaluation (PROMETHEE) (Behzadian et al. 2010) method, Superiority and Inferiority Ranking
(SIR)method (Tam et al 2004), Multi Attribute Global Inference Quality (MAGIQ)method (MCaffrey and James 2005), Analytical Hierarchy Process (AHP) etc. One such evaluation method is to use any of the techniques that belong to the family of Multi Criteria Decision Analysis (MCDA) methods (Morisio and Tsoukios 1997). As an example, Kontio (1996) discusses using AHP (Analytic Hierarchy Process) for evaluating the reusability of COTS components.

Let us illustrate the two popular methods of WSM and WPM with an example. Let the given MCDA problem be defined on three alternatives and four decision criteria. It is assumed that all the criteria are benefit criteria, that is the higher the values the better it is. The weights assigned to the criteria based on their importance and their performance values corresponding to the alternatives are given in Table 1.1.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight</td>
<td>0.1</td>
<td>0.3</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>Alternative 1</td>
<td>20</td>
<td>40</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>Alternative 2</td>
<td>30</td>
<td>20</td>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td>Alternative 3</td>
<td>40</td>
<td>30</td>
<td>20</td>
<td>30</td>
</tr>
</tbody>
</table>

1.12.1 Weighted Sum Model (WSM)

The Weighted Sum model (WSM) is considered first. This method is applicable only when all the data are expressed in exactly the same unit. If this were not the case one would be guilty of adding “Apples to Oranges”. As the name suggests, total performance is obtained by summing up the performances corresponding to all the criteria for the chosen alternative.
The performance scores for three alternatives are as follows:

\[
\begin{align*}
A1 & : 20 \times 0.1 + 40 \times 0.3 + 30 \times 0.4 + 10 \times 0.2 = 2 + 12 + 12 + 2 = 28 \quad (1.1) \\
A2 & : 30 \times 0.1 + 20 \times 0.3 + 40 \times 0.4 + 20 \times 0.2 = 3 + 6 + 16 + 4 = 29 \quad (1.2) \\
A3 & : 40 \times 0.1 + 30 \times 0.3 + 20 \times 0.4 + 30 \times 0.2 = 4 + 9 + 8 + 6 = 27 \quad (1.3)
\end{align*}
\]

Thus the most profitable alternative is A2 followed by A1 and alternative A3 gets the third place.

### 1.12.1.1 The Weighted Product Model (WPM)

The weighted product model is another popular method. It is similar to the weighted sum model. It has the advantage that the performance criteria can be expressed in different units. The main difference between WSM and WPM is that in WPM multiplicative operators obtain the cumulative score.

Averting to the example the performance scores of the three alternatives are evaluated as:

\[
\begin{align*}
(A1/A2) & = (20/30)^{0.1} \times (40/20)^{0.3} \\
& \times (30/40)^{0.4} \times (10/20)^{0.2} = 0.959 \times 1.23 \times 0.891 \times 0.87 \\
& = 0.914 \quad (1.4) \\
(A1/A3) & = (20/40)^{0.1} \times (40/30)^{0.3} \times (30/20)^{0.4} \times (10/30)^{0.2} \\
& = 0.933 \times 1.089 \times 1.176 \times 0.802 = 0.959 \quad (1.5) \\
(A2/A3) & = (30/40)^{0.1} \times (20/30)^{0.3} \times (40/20)^{0.4} \times (20/30)^{0.2} \\
& = 0.971 \times 0.885 \times 1.319 \times 0.922 = 1.045 \quad (1.6)
\end{align*}
\]

There are scores of MCDA methods available today. Further new methods are being added routinely. However, different methods may yield different results for exactly the same problem. In other words, when exactly the same problem data is used with different MCDA methods, each method
may recommend different solutions for the same problem. This raises the fundamental issues of how to evaluate and compare various MCDA methods (Triantaphyllou 2000). Choosing the best MCDA / MCDM method is itself a multi-criteria decision making problem, in which the alternatives are the methods themselves and the decision criteria are the various evaluative ways for comparing them. It is important to choose the best MCDA method (Triantaphyllou 1989).

In this thesis the Weighted Average Sum (WAS) model, a variation of the Weighted Sum Model (WSM) is used. The Weighted Average Sum (WAS) is the most well known and simplest MCDA method for evaluating a number of alternatives in terms of a number of decision criteria. It is used in most everyday evaluations (Morisio and Tsoukios 1997). It is also the most preferred method used whenever an ordinal scale (very high, high, low) is transformed into a ratio (3,2,1) (Morisio and Tsoukios 1997). The main objective of the thesis is to create a basic quality model that can be used for COTS selection. A minimal set of attributes have been proposed and have been validated by a sample group of twenty professionals. It has been found that for the given set of attributes and the experimental data set the Weighted Average Sum Model (WAS) is the best. However, it is stressed that the proposed quality model must be tested by other MCDA methods with a larger group of respondents and successively refined, before it can be accepted as a global standard.

1.13 ORGANIZATION OF THE THESIS

Chapter 2 of the thesis presents a study of the existing quality models in component research literature. A comparison is carried out between the different models and the advantages and drawbacks of the different models have been discussed.
Chapter 3 presents the TRUSTAD Nomenclature that can be used to specify and also distinguish between COTS, in-house and reusable components. The chapter presents formal definitions for COTS, in-house and reusable components.

Chapter 4 provides the outline for the Q’Facto 10 quality model that can be used for the evaluation of COTS components only. This model was built using the definitions and specification for COTS components presented in chapter 3. The model is called the Q’Facto 10 model because it has ten quality factors and is based on the ISO 9126 quality standard.

Chapter 5 presents a refined model of the previous Q’Facto 10 model called the Q’Facto 12 model that has twelve high level quality factors and uses the ISO 25000 [SQuaRE] quality standard as a baseline.

Chapter 6 describes the model validation and presents the results of the model validation that has been done using statistical analysis.

Chapter 7 discusses the future work and enhancements that can be done to the existing work.

1.14 CONCLUSION

In this chapter an introduction to COTS software components was provided. The need for a COTS quality evaluation standard was highlighted. The difficulties in developing such a standard were discussed. The different ways in which these issues could be overcome were mentioned. The software quality perspective has been discussed. The different software quality models were presented and the advantages of the ISO 9126 model were discussed. Finally, a brief description of the organization of the thesis has been given. The next chapter in continuation presents the literature survey.