



Pós-Graduação em Ciência da Computação

“A Software Component Quality Framework”

By

Alexandre Alvaro

Ph.D THESIS



Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, 05/2009



UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ALEXANDRE ALVARO

"A Software Component Quality Framework"

ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIA DA COMPUTAÇÃO.

A PHD. THESIS PRESENTED TO THE FEDERAL UNIVERSITY OF PERNAMBUCO IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF PHD IN COMPUTER SCIENCE.

ADVISOR: Silvio Romero de Lemos Meira

RECIFE, MAY/2009

Álvaro, Alexandre

**A software component quality framework / Alexandre
Alvaro. – Recife : O Autor, 2009.**

xv, 208 folhas : il., fig., tab.

**Tese (doutorado) – Universidade Federal de
Pernambuco. Cln. Ciência da Computação, 2009.**

Inclui bibliografia e apêndices

**1. Componentes de software - Qualidade. 2. Modelo de
qualidade de componentes. 3. Técnicas e processos de
avaliação de componentes. I. Título.**

**004
005**

**CDU (2.ed.)
CDD (22.ed.)**

**UFPE
3C - 2009 - 098**

*To my wonderful and well-beloved family:
Valdir and Maria
Denise, Fabio, Eduardo and Murilo
And to the love of my life:
Mariana*



cknowledgements

This thesis embodies a large effort on my part, but it could not have been developed without the support of many people. I would like to thank all those who have helped me along this journey, knowing that I will never be able to truly express my gratefulness.

First of all, I would like to thank my advisor, Silvio Romero de Lemos Meira, who always gave me support during this long journey. He is an incredible and open-minded person who I had the opportunity of sharing some (a lot of) ideas about the most important topics about computer science besides the pleasure of drinking some beers and talking in weekends. Moreover, he always gave me support during my return for my hometown, believing that I could finish my PhD even with the distance. I must repeat the same phrase that I put in the acknowledgements of my Master degree: *“He is the advisor that all post-graduate students would like to work with”*. Thank you, a lot!!

In particular, I would like to thank my Thesis members, Alexandre Vasconcelos, Hermano Perrelli de Moura, Andre Luis Medeiros Santos, Renata Pontin de Mattos Fortes and Ricardo Falbo for the time input and many stimulating discussions. Their comments greatly helped to improve my thesis.

I am thankful, again, to Alexandre Vasconcelos, for giving me insights in software quality, which helped in the definition of the component quality framework, and for always being available to read my papers. In addition, he gave me the contact of Danilo Scalet who indicated me to participate of the software quality committee in Brazil (ABNT NBR ISO/IEC 25000).

I am deeply grateful to all of my friends that shared some moments with me. Thanks Ricardo Alexandre Afonso (Alonso) for his company during this

period in Recife. He is a very comic person that aid me to support living far from my family. All the parties, drinks, travelling, jobs, time to talk about the life, about our future, the Saturdays in the Boa Viagem beach, among other things that we done were very special for my life. Of course, my life in Recife became better due to his company. In the same way, thanks also Ricardo Argenton Ramos (RAR) for his company in Recife and for all wonderful lunches we did together.

Thanks for my friends at Sorocaba-SP (Bruno and Neto). Even with the distance, they were always available for talking and giving me force for continuing this work. Moreover, thanks to my friends Fabio Ogawa and Cleber, who are always interested in my future ways, in order to remember together the old times and put in practice the “dreams” of the university. I am very thankful to all my friends that always listen to me, my questions about life, the future and everything.

I also want to thanks to the entire RiSE group for their support in this whole time. Particularly Eduardo Almeida and Vinicius Garcia, the person who I lived and worked for a long period. They were always available to help me in some circumstances. Thanks Daniel Lucrédio for all of his attention during all of this work. Its support in all the time – in the English, in the papers, in the future – was very important to me. Thanks also for the careful revision of this thesis.

Finally, I would like to thank C.E.S.A.R. (Recife Center for Advanced Studies and Systems), and again Silvio Meira, for providing me the means to put my work in a real practice environment. And to the Informatics Center at UFPE for making available its infrastructure.

And of course, I offer very special thanks to my beloved family. My mom and papa for their happiness, love and company in some difficult moments. I would like to thank them for being always available with constant support in the hardest situations, when I needed them the most, even so distant. Sorry for staying so far during this long time, the homesickness was enormous and for our happiness I came back. So, I have a piece of music for you two:

*“As vezes é tormenta,
Fosse uma navegação.
Pode ser que o barco vire
Também pode ser que não*

*Já dei meia volta ao mundo
Levitando de tesão
Tanto gozo e sussurro
Já impressos no colchão...*

*Pois sempre tem
A cama pronta
E rango no fogão, fogão...*

*Luz acesa
Me espera no portão
Pra você ver
Que eu tô voltando pra casa
E vê! ê! ê! ê! ê!
Que eu tô voltando pra casa
Outra vez..."*

(Casa – Lulu Santos)

Mom and papa, I love you two, so much!!

Grateful thanks to my lovely sister, Denise, who was always present in all days that I was distant and always trying to decrease the distance between my hometown and Recife. Her support in some moments during this period was fundamental for me. Now, I´m back and share the good moments together with her and her family. Thanks for Deba, I love you. Very special thanks to Fabio, for his company in some moments in Sorocaba/SP, for all barbecues, beers, joking and talking. Thanks Fabão. Thanks for my friend Eduardo, he is a very sweet and smart children that I love playing with him. He touches my heart some times when saying for me don´t go far away and stay with him to play. Now, I´m back. I love you Du!! Thanks for the newer people of my family, Murilo, the most beautiful and likeable baby that I know.

A special thanks for Mariana, who tried to be patient during my journey in Recife since 2004. Every single moment far from her love was very hard for me. She is the person that gives me support, motivation and inspiration to continue my work until the end (since so many times were not so easy doing so). Thanks a lot for understanding and comprehending the difficulties that we passed during this time. I hope to be with you all of my life. I love you so much my dear!!

So, they are the most important people in all of my life. I hope for that one day I will give them as much as they have given to me.

I am so grateful for all the people that contributed with something during the development of this work. Mainly, thanks for my Mom, my Papa, my sister Deba, my friends Fabão, Eduardo, Murilo baby and my love Mariana. Without them, I would not be where I am now. They will always be in my heart!!

Ando devagar porque já tive pressa
E levo esse sorriso porque já chorei demais
Hoje me sinto mais forte, mais feliz quem sabe
Só levo a certeza de que muito pouco eu sei
Ou nada sei

Conhecer as manhas e as manhãs,
O sabor das massas e das maçãs,
É preciso amor pra poder pulsar,
É preciso paz pra poder sorrir,
É preciso a chuva para florir

Todo mundo ama um dia todo mundo chora,
Um dia a gente chega, no outro vai embora
Cada um de nós compõe a sua história
Cada ser em si carrega o dom de ser capaz
E ser feliz

Conhecer as manhas e as manhãs,
O sabor das massas e das maçãs,
É pra poder pulsar,
É preciso paz pra poder sorrir,
É preciso a chuva para florir

Ando devagar porque já tive pressa
E levo esse sorriso porque já chorei demais
Cada um de nós compõe a sua história,
Cada ser em si carrega o dom de ser capaz
E ser feliz

(Tocando em Frente, Renato Teixeira)



Resumo

Um grande desafio da Engenharia de Software Baseada em Componentes (ESBC) é a qualidade dos componentes utilizados em um sistema. A confiabilidade de um sistema baseado em componentes depende da confiabilidade dos componentes dos quais ele é composto. Na ESBC, a busca, seleção e avaliação de componentes de software é considerado um ponto chave para o efetivo desenvolvimento de sistemas baseado em componentes. Até agora a indústria de software tem se concentrado nos aspectos funcionais dos componentes de software, deixando de lado uma das tarefas mais árduas, que é a avaliação de sua qualidade. Se a garantia de qualidade de componentes desenvolvidos *in-house* é uma tarefa custosa, a garantia da qualidade utilizando componentes desenvolvidos externamente – os quais frequentemente não se tem acesso ao código fonte e documentação detalhada – se torna um desafio ainda maior. Assim, esta Tese introduz um *Framework* de Qualidade de Componentes de Software, baseado em módulos bem definidos que se complementam a fim de garantir a qualidade dos componentes de software. Por fim, um estudo experimental foi desenvolvido e executado de modo que se possa analisar a viabilidade do *framework* proposto.

Palavras Chave: Componentes de Software, Qualidade de Componentes de Software, Modelo de Qualidade de Componentes, Técnicas para Avaliação de Componentes e Processo de Avaliação de Componentes.



Abstract

A major problem with Component-Based Software Engineering (CBSE) is the quality of the components used in a system. The reliability of a component-based software system depends on the reliability of the components that it is made of. In CBSE, the proper search, selection and evaluation process of components is considered the cornerstone for the development of any effective component-based system. So far the software industry was concentrated on the functional aspects of components, leaving aside the difficult task of assessing their quality. If the quality assurance of in-house developed software is a demanding task, doing it with software developed elsewhere, often without having access to its source code and detailed documentation, presents an even greater concern. So, this Thesis introduces a Software Component Quality Framework, based upon well-defined modules that complement each other looking for assurance the component quality in a consistent way. An experimental study was developed and executed in order to analyze the viability of using such a framework.

Keywords: Software Components, Software Component Quality, Component Quality Model, Component Evaluation Techniques and Component Evaluation Process.



ist of Figures

FIGURE 1.1 THE FRAMEWORK FOR SOFTWARE REUSE.....	03
FIGURE 1.2 SOFTWARE COMPONENT QUALITY FRAMEWORK.....	04
FIGURE 1.3 SUMMARY OF SURVEY RESPONSE (BASS ET AL., 2000).....	05
FIGURE 3.1 NUMBER OF COMPANIES CERTIFIED ISO 9000, CMMI AND MPS.br.....	26
FIGURE 3.2 SQUARE ARCHITECTURE (ISO/IEC 25000, 2005).....	27
FIGURE 3.3 ISO/IEC 25040.....	32
FIGURE 3.4 PROCESS OF OBTAINING, CERTIFYING AND STORING COMPONENTS.....	35
FIGURE 4.1 STRUCTURE OF PREDICTION-ENABLED COMPONENT TECHNOLOGY (WALLNAU, 2003).....	47
FIGURE 4.2 RESEARCH ON SOFTWARE COMPONENT CERTIFICATION TIMELINE.....	50
FIGURE 5.1 SOFTWARE COMPONENT QUALITY FRAMEWORK.....	56
FIGURE 5.2 RELATIONS AMONG THE QUALITY MODEL ELEMENTS.....	64
FIGURE 5.3 SUMMARY OF THE CQM.....	74
FIGURE 7.1 COMPONENT EVALUATION PROCESS.....	99
FIGURE 7.2 ESTABLISH EVALUATION REQUIREMENTS STEPS.....	100
FIGURE 7.3 SPECIFY THE EVALUATION STEPS.....	105
FIGURE 7.4 DESIGN THE EVALUATION STEPS.....	109
FIGURE 7.5 EXECUTE THE EVALUATION STEPS.....	113
FIGURE 8.1 SUBJECT'S TIME SPENT IN THE EXPERIMENTAL STUDY.....	125
FIGURE 8.2 COMPONENT QUALITY MEASURED: PERSISTENCE MANAGER AND ARTIFACT MANAGER.....	130
FIGURE 8.3 COMPONENT QUALITY MEASURED: ASSET SEARCHER, ASSET CATALOG AND INDEXER.....	132



ist of Tables

TABLE 3.1	SOFTWARE QUALITY STANDARDS.....	25
TABLE 3.2	CHARACTERISTICS AND SUB-CHARACTERISTICS IN SQUARE PROJECT.....	31
TABLE 5.1	CHANGES IN THE PROPOSED COMPONENT QUALITY MODEL, IN RELATION TO ISO/IEC 25010.....	58
TABLE 5.2	THE PROPOSED COMPONENT QUALITY MODEL, WITH THE SUB-CHARACTERISTICS BEING DIVIDED INTO TWO KINDS: RUNTIME AND DEVELOPMENT TIME.....	63
TABLE 5.3	COMPONENT QUALITY ATTRIBUTES FOR SUB-CHARACTERISTICS THAT ARE OBSERVABLE AT <i>RUNTIME</i>	65
TABLE 5.4	COMPONENT QUALITY ATTRIBUTES FOR SUB-CHARACTERISTICS THAT ARE OBSERVABLE DURING <i>LIFE CYCLE</i> ...	67
TABLE 5.5	ADDITIONAL INFORMATION.....	73
TABLE 6.1	GUIDELINES FOR SELECTING EVALUATION LEVEL.....	77
TABLE 6.2	SOFTWARE COMPONENT TECHNIQUES MODEL.....	86
TABLE 6.3a	COMPONENT QUALITY ATTRIBUTES X EVALUATION TECHNIQUES.....	87
TABLE 6.3b	COMPONENT QUALITY ATTRIBUTES X EVALUATION TECHNIQUES.....	88
TABLE 6.3c	COMPONENT QUALITY ATTRIBUTES X EVALUATION TECHNIQUES.....	89
TABLE 7.1	EXAMPLE OF IMPORTANCE'S DEFINITION.....	103
TABLE 7.2	EXAMPLE OF QUALITY ATTRIBUTES DEFINITION.....	105
TABLE 7.3	EXAMPLE OF DEFINE SCTM.....	106
TABLE 7.4	EXAMPLE OF TOOLS SELECTION.....	111
TABLE 7.5	EXAMPLE OF DATA COLLECTION.....	114
TABLE 8.1	SUBJECT'S PROFILE IN THE EXPERIMENTAL STUDY.....	126
TABLE 8.2	QUALITY ATTRIBUTES SELECTED FOR SCTM I.....	127
TABLE 8.3	QUALITY ATTRIBUTES SELECTED FOR SCTM II.....	127
TABLE 8.4	EVALUATION TECHNIQUES SELECTED TO THE COMPONENTS EVALUATED IN SCTM I.....	128
TABLE 8.5	EVALUATION TECHNIQUES SELECTED TO THE COMPONENTS EVALUATED IN SCMT II.....	130



cronyms

Term	Description
B2B	Business to Business
CBD	Component-Based Development
CBSE	Component-Based Software Engineering
CMU/SEI	Carnegie Mellon University's Software Engineering Institute
COTS	Commercial Off-The-Self
CBSD	Component-Based Software Development
CQM	Component Quality Model
COM	Component Object Model
CCM	CORBA Component Model
CMMI	Capability Maturity Model Integrated
EJB	Enterprise JavaBeans
GQM	Goal Question Metric Paradigm
ISO	International Organization for Standardization
IEC	International Electro-technical Commission
PECT	Prediction-Enabled Component Technology
PACC	Predictable Assembly from Certifiable Components
RiSE	Reuse in Software Engineering group
SCETM	Software Component Evaluation Techniques Model
SQuaRE	Software Product Quality Requirements and Evaluation
OMG	Object Management Group
XML	eXtensible Markup Language



ontents

1. INTRODUCTION	1
1.1 MOTIVATION	1
1.1.1 <i>Software Components Inhibitors</i>	4
1.1.2 <i>The Future of Software Components</i>	6
1.1.3 <i>The Brazilian Industry Relevance</i>	7
1.2 PROBLEM STATEMENT.....	8
1.3 OVERVIEW OF THE PROPOSED SOLUTION	9
1.4 STATEMENT OF THE CONTRIBUTIONS	10
1.5 OUT OF SCOPE	11
1.6 ORGANIZATION OF THE THESIS	12
2. SOFTWARE REUSE	15
2.1 COMPONENT-BASED DEVELOPMENT (CBD).....	19
2.1.1 <i>Software Components</i>	20
2.2 SUMMARY	22
3. SOFTWARE QUALITY AND CERTIFICATION	23
3.1 ISO/IEC 25000 (SQUARE PROJECT)	26
3.2.1 ISO/IEC 2501N (QUALITY MODEL DIVISION)	30
3.2.2 ISO/IEC 2504N (QUALITY EVALUATION DIVISION).....	32
3.2.3 ISO/IEC 2502N (QUALITY MEASUREMENT DIVISION)	33
3.3 SOFTWARE COMPONENT CERTIFICATION	33
3.4 SUMMARY	36
4. SOFTWARE COMPONENT CERTIFICATION: A SURVEY	37
4.1 EARLY AGE: MATHEMATICAL AND TEST-BASED MODELS	38
4.2 SECOND AGE: TESTING IS NOT ENOUGH TO ASSURE COMPONENT QUALITY...43	
4.3 FAILURES IN SOFTWARE COMPONENT CERTIFICATION.....	49
4.4 CONCLUSION OF THE STUDY	50
4.5 SUMMARY	51
5. SOFTWARE COMPONENT QUALITY FRAMEWORK AND COMPONENT QUALITY MODEL	52
5.1 OVERVIEW OF THE FRAMEWORK.....	53
5.2 THE COMPONENT QUALITY MODEL (CQM)	56
5.2.1 CHANGES IN RELATION TO ISO/IEC 25010	57
5.2.2 QUALITY CHARACTERISTICS THAT WERE ADAPTED FROM ISO/IEC 25010 ..	61

5.2.3	SUMMARY	63
5.3	COMPONENT QUALITY ATTRIBUTES	64
5.4	OTHER RELEVANT COMPONENT INFORMATION	73
5.5	SUMMARY	73
6.	EVALUATION TECHNIQUES FRAMEWORK AND METRICS FRAMEWORK	75
6.1	A SOFTWARE COMPONENT MATURITY MODEL (SCTM).....	76
6.2	METRICS FRAMEWORK.....	91
6.2.1	METRICS TO TRACK THE CQM PROPERTIES	93
6.2.2	METRICS TO TRACK THE CERTIFICATION TECHNIQUES PROPERTIES	95
6.2.3	METRICS TO TRACK THE CERTIFICATION PROCESS PROPERTIES.....	96
6.3	SUMMARY	97
7.	COMPONENT EVALUATION PROCESS	98
7.1	THE COMPONENT EVALUATION PROCESS	98
7.1.1	ESTABLISH EVALUATION REQUIREMENTS ACTIVITY	99
7.1.2	SPECIFY THE EVALUATION ACTIVITY.....	104
7.1.3	DESIGN THE EVALUATION ACTIVITY	109
7.1.4	EXECUTE THE EVALUATION ACTIVITY.....	112
7.1.5	PROCESS SUMMARY.....	115
7.2	SUMMARY	115
8.	EXPERIMENTAL STUDY	116
8.1	SOFTWARE COMPONENT QUALITY FRAMEWORK – AN EXPERIMENTAL STUDY 117	
8.2	DEFINITION OF THE EXPERIMENTAL STUDY	117
8.3	PLANNING OF THE EXPERIMENTAL STUDY.....	117
8.4	THE PROJECT USED IN THE EXPERIMENTAL STUDY	123
8.5	THE INSTRUMENTATION.....	123
8.6	THE OPERATION	123
8.7	THE ANALYSIS AND INTERPRETATION	126
8.8	LESSONS LEARNED.....	135
8.9	SUMMARY	135
9.	CONCLUSIONS	137
9.1	RESEARCH CONTRIBUTIONS	138
9.2	RELATED WORK.....	139
9.3	FUTURE WORK	139
9.4	ACADEMIC CONTRIBUTIONS.....	140
9.5	OTHER PUBLICATIONS.....	144
9.6	SUMMARY	145
	REFERENCES.....	147
	APPENDIX A. METRICS EXAMPLE	167
	APPENDIX B. COMPONENT QUALITY EVALUATION FORM	188
	APPENDIX C. QUESTIONNAIRES USED IN THE EXPERIMENTAL STUDY.....	200



Introduction

1.1 Motivation

One of the most compelling reasons for adopting component-based approaches in software development is the premise of reuse. The idea is to build software from existing components primarily by assembling and replacing interoperable parts. The implications for reduced development time and improved product quality make this approach very attractive (Krueger, 1992).

Reuse is a “generic” denomination, encompassing a variety of techniques aimed at getting the most from design and implementation work. The top objective is to avoid reinvention, redesign and reimplementation when building a new product, by capitalizing on previous done work that can be immediately deployed in new contexts. Therefore, better products can be delivered in shorter times, maintenance costs are reduced because an improvement to one piece of design work will enhance all the projects in which it is used, and quality should improve because reused components have been well tested (D’Souza et al., 1999), (Jacobson et al., 1997).

Software reuse is not a new idea. Since McIlroy’s pioneer work, “*Mass Produced Software Components*” (McIlroy, 1968), the idea of reusing software components in large scale is being pursued by developers and research groups. This effort is reflected in the literature, which is very rich in this particular area of software engineering.

Most of this works follow McIlroy’s idea: “*the software industry is weakly founded and one aspect of this weakness is the absence of a software component sub-industry*” (pp. 80). The existence of a market, in which

developers could obtain components and assemble them into applications, was always envisioned.

The influence of McIlroy's work has led the research in creation of component repository systems, involving complex mechanisms to store, search and retrieve assets. This can be seen in a software reuse libraries survey (Mili et al., 1998), where Mili et al. discuss about 50 works that deal with the reuse problem.

On the other hand, these works do not consider an essential requirement for these systems: the assets quality. In a real environment, a developer that retrieves a faulty component from the repository would certainly lose his trust on the system, becoming discouraged to make new queries. Thus, it is extremely important to assert the quality of the assets that are stored into the repository before making them available for reuse. Despite this importance, the software engineering community had not explored these issues until recently. In this way, a new research area arose: components quality assurance (Wohlin et al., 1994), (Mingins et al., 1998), (Morris et al., 2001), (Schmidt, 2003), (Wallnau, 2003). However, several questions still remain unanswered, such as: **(i)** how component quality assurance should be carried out? **(ii)** what are the requirements for a component evaluation process? and, **(iii)** who should perform it? (Goulão et al., 2002a). This is the reason why there is still no well-defined standard to perform component quality assurance (Voas et al., 2000), (Morris et al., 2001).

In this context, the main goal of this thesis is investigating effective ways to demonstrate that component quality assurance is practically viable to researcher and software/quality engineer. Through component quality, some benefits can be achieved, such as: higher quality levels, reduced maintenance time, investment return, reduced time-to-market, among others. According to Weber et al. (Weber et al., 2002), the need for quality assurance in software development has exponentially increased in the past few years.

Moreover, this thesis is part of a bigger project whose main idea is to develop a robust framework for software reuse (Almeida et al., 2004), in order to establish a standard for component development; to develop a repository system; and to develop a component quality framework. This project has been

developed in a collaboration between the industry and academia (the RiSE group¹ and two universities), in order to generate a well-defined model for developing, evaluating quality, storing and, after that, making it possible for software factories to reuse software components.

The framework (Figure 1.1) that is being developed has two modules. The first module (on the left) is composed of best practices related to software reuse. Non-technical factors, such as education, training, incentives, and organizational management are considered. This module constitutes a fundamental step before the introduction of the framework in the organization. The second module (on the right) is composed of important technical aspects related to software reuse, such as processes, environments, tools, and a component quality framework, which is the focus of this thesis.

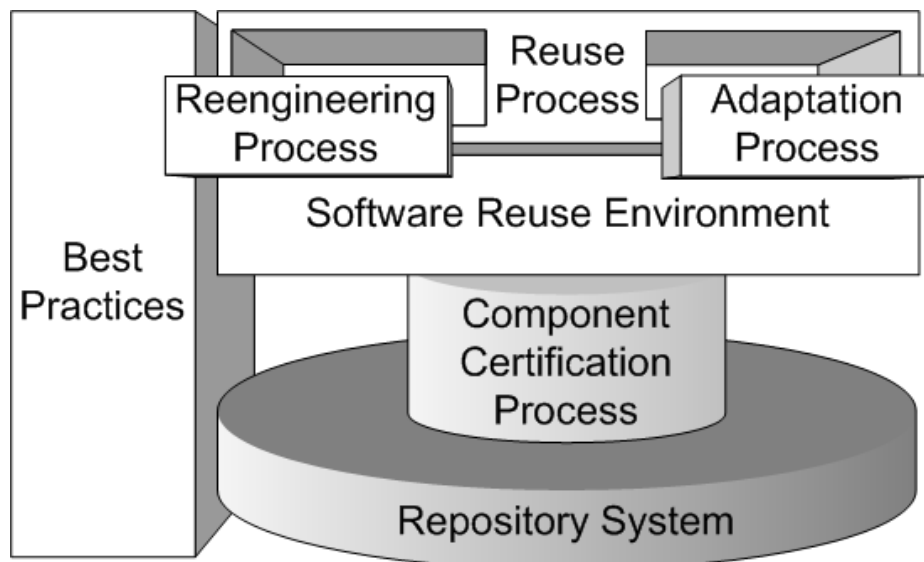


Figure 1.1. The Framework for Software Reuse.

The process of evaluation components quality is not a simple one. First, there should exist a **component quality model**. Differently from other software product quality models, such as (McCall et al., 1977), (Boehm et al., 1978), (Hyatt et al., 1996), (ISO/IEC 9126, 2001), (Georgiadou, 2003), this model should consider Component-Based Development (CBD) characteristics, and describe attributes that are specific to the promotion of reuse². With a

¹ Reuse in Software Engineering (RiSE) group – <http://www.rise.com.br>.

² The component quality model was developed as part of my MSc. degree in computer science and upgraded during this thesis.

component quality model in hand, there must be a series of **techniques** that allow one to evaluate if a component conforms to the model. The correct usage of these techniques should follow a well-defined and controllable **component evaluation process**. Finally, a set of **metrics** are needed, in order to track the components properties and the enactment of the evaluation process.

These four main issues: **(i)** a Component Quality Model, **(ii)** a Quality Techniques Framework, **(iii)** a Metrics Framework, and **(iv)** a Evaluation Process, are the modules of a Software Component Quality Framework that is being investigated as a part of the RiSE project.

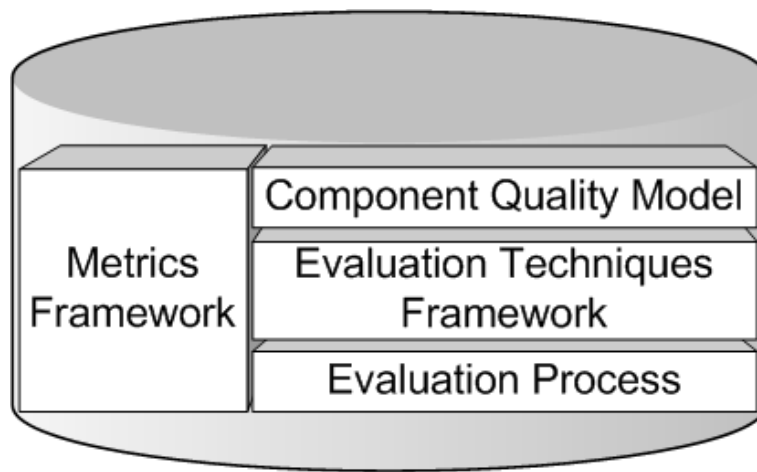


Figure 1.2. Software Component Quality Framework.

The framework will allow that the components produced in a *Software Reuse Environment* are certified before being stored on a *Repository System*. In this way, software engineers would have a greater degree of trust in the components that are being reused.

1.1.1 Software Components Inhibitors

To assess the market for Component-Based Software Engineering (CBSE), the Carnegie Mellon University's Software Engineering Institute (CMU/SEI) studied industry trends in the use of software components. The study (Bass et al., 2000), conducted from September 1999 to February 2000, examined software components from both technical and business perspectives.

A distinct set of inhibitors to adopting software component technology emerged from the conducted surveys and interviews with earlier adopters of

software component technology. A summary from a Web survey of component adopters is presented in Figure 1.3.

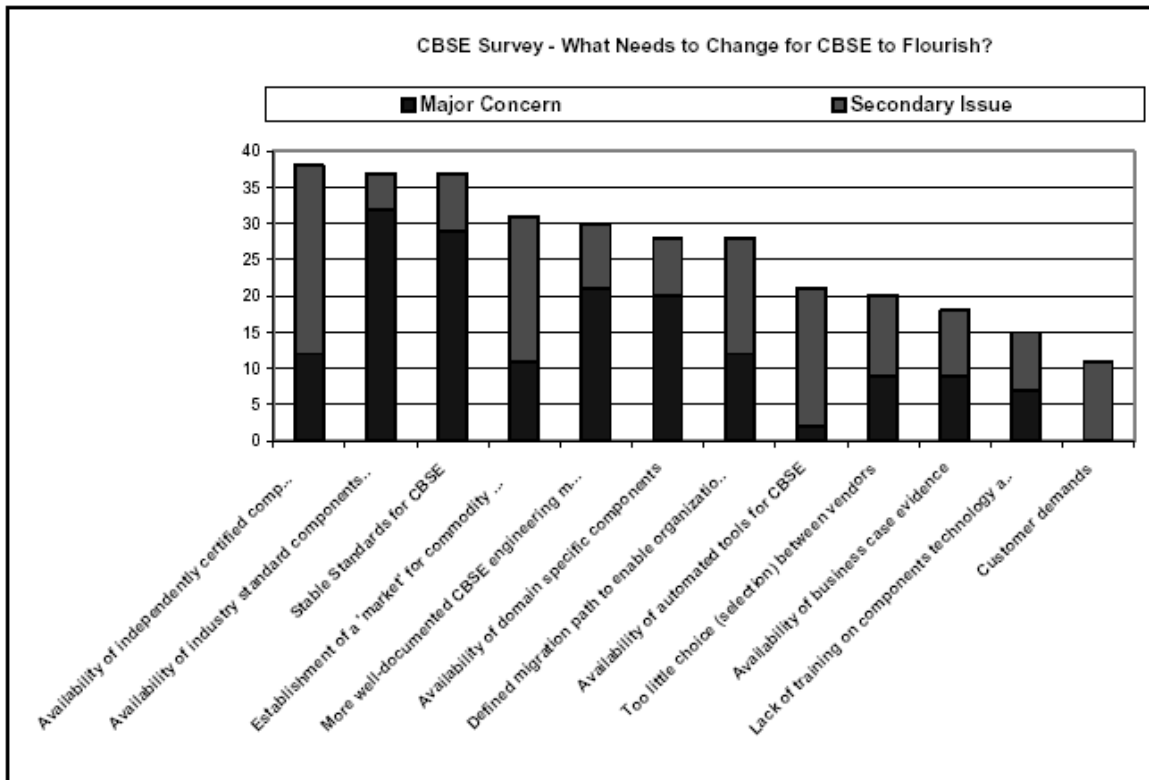


Figure 1.3. Summary of Survey Responses (Bass et al., 2000).

From this data and from the interviews, the study concluded that the market perceives the following key inhibitors for component adoption, presented here in decreasing order of importance:

- Lack of available components;
- Lack of stable standards for component technology;
- **Lack of certified components;** and
- Lack of an engineering method to consistently produce quality systems from components.

The software engineering community is already attempting to reduce the gaps related to the two first inhibitors. A look at the main Internet component market, *ComponentSource*³, which contains more than 5,000 components (seven years ago it had less than 1,000 (Trass et al., 2000)), leads to conclude that there is a large increase in the availability of reusable assets. In the same period, component technologies have obtained considerable maturity, especially

³ <http://www.componentsource.com>

those related to JavaBeans, Enterprise JavaBeans (EJB), and Microsoft .NET technologies. Thus, the software engineering community is trying to establish stable standards for component technology, each one for a particular market niche.

However, in relation to the third inhibitor, the community is still a fledgling. Further research is required in order to assure the production of certified components, especially when combined with the lack of component-based software engineering techniques that deliver predictable properties (the last inhibitor).

The concern with components quality assurance reflects a natural progression of concerns: first demonstrates that it is possible to build and sustain a component-based system at all, and then improve the overall quality of components and the consumers' trust in these components.

According to Voas (Voas, 1998), to foster an emerging software component marketplace, it must be clear for buyers whether a component's impact is positive or negative. Ideally, buyers should have this information before buying a component. Component buyers could then choose an appropriate component according to its quality level. With this information, system builders could make better design decisions and be less fearful of liability concerns, and component vendors could expect a growing marketplace for their products.

1.1.2 The Future of Software Components

Important researches on software components, such as (Heineman et al., 2001), (Heineman et al., 2001), (Crnkovic, 2001), (Wallnau, 2003), (Wallnau, 2004), (Schneider & Han, 2004) and (Andreou & Tziakouris, 2007) point that the future of software components is quality assurance. These authors state that quality assurance is a necessary precondition for CBSE to be successfully adopted and to achieve the associated economic and social benefits that CBSE could yield. With the success of CBSE, software developers will have more time to develop, instead of spending their time addressing problems associated with understanding and fixing someone else's code. Components with quality used during development will have predetermined and well-established criteria, thus

reducing the risk of system failure and increasing the likelihood that the system will comply with design standards.

When the system is developed using a CBSE approach, the use of components with quality could provide objective evidence that the components meet rigorous specifications including data on intended use. This approach does not permit the designer to forego inherently safe system design practices. Instead, quality procedures reduce the risk of system failure by providing information about a software component risk mitigation rules, such as the anticipation about the software failure state and return to the last stable state with notice to the system administrator. The objective is to build safe systems from well-documented and proven components. And if these components are independently assessed, the confidence that the information accompanying these components meets their requirements will be greater.

1.1.3 The Brazilian Industry Relevance

In 2007 it was accomplished a study by SOFTEX (Softex, 2007) in conjunction with the *Departamento de Política Científica e Tecnológica da Unicamp* and financed by *Ministério de Ciência e Tecnologia (MCT)*, with the idea of evaluating the main contribution that CBSE can bring to the Brazilian Software Industry. A set of CBD specialists discussed technical and economical aspects related to the software component adoption by the national industry and defined the main goals to achieve it, namely:

- **Quality of the software developed;**
- Maintainability;
- **Reliability;** and
- Service-Oriented Architecture (SOA) evolution.

According to those specialists, the **quality** and **reliability** of the components developed and availability in the market are the main aspects to increase the component adoption degree in Brazilian organizations. One of the interesting aspects pointed out is the fact that reusing a component without quality could be worse than not reusing anything.

Additionally, a survey involving 57 Brazilian small, medium and large software organizations was accomplished (Lucrédio et al., 2007) and it could be noted that the quality of the components available in the market is a critical factor for the software reuse success like other aspects: systematic reuse process, CASE tools usage, product family approach, independent reusable assets development team, among others. Basically, this study presented the same line of reasoning of the previously study, where an asset without quality could be a risk-factor to the software development project.

In this way, according to SEI (Bass et al., 2003), Softex (Softex, 2007) and Brazilian software organizations (Lucrédio et al., 2007), it was considering that the component quality are an essential aspect to the CBSE adoption and software reuse success around the world.

1.2 Problem Statement

The growing use of commercial components in large systems makes selection and evaluation of components an essential activity. Many organizations struggle in their attempts to select and evaluate an appropriate component in their system. For this reason, a well-defined and consistent software component quality assurance is essential for the component market adoption (i.e. without a efficient mechanism to select/evaluate the component quality, the organization will often select components with low quality and, therefore, it will be discouraged to select other components in the future and would certainly lose its trust on the component market).

According to Morris et al. (Morris et al., 2001), there is a lack of an effective assessment of software components. Besides, the international standards that addresses software products' quality issues have shown to be too general for dealing with the specific characteristics of components. While some of their characteristics are appropriate to the evaluation of components, others are not well suited for that task.

Even so, the software engineering community has expressed many and often diverging requirements to CBSE and trustworthy components. A unified and prioritized set of CBSE requirements for trustworthy components is a challenge in itself (Goulão et al., 2002a), (Schmidt, 2003).

Moreover, there is a lack of processes, methods, techniques and tools that support the component quality assurance activity (Alvaro et al., 2005a). The current processes and methods only deal with specific aspects of software component (Alvaro et al., 2005a) and were not evaluated into industrial scenarios. In fact, they are based on the researchers' experience, and the real efficiency of evaluating software components using these process/methods remains unknown.

In this way, the target problem that this thesis intends to work with is the lack of mechanisms available on the literature for evaluates the software component quality degree, providing consistent and well-define framework for software component quality evaluation.

1.3 Overview of the Proposed Solution

The component market, which is a priori condition to maximize the intra- and inter-organizational software reusability, cannot emerge without supplying high-quality products. Organizations whose aim is 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 satisfy certain functional and quality requirements. Without a quality level, the component usage may have catastrophic results (Jezequel et al., 1997). However, the common belief is that the market components are not reliable and this prevents the emergence of a mature software component market (Trass et al., 2000). Thus, the components market quality problems must be resolved in order to increase the reliability, and third-party quality assurance programs would help to acquire trust in the market components (Heineman et al., 2001).

Motivated by these ideas, the main goal of this thesis is to define a Software Component Quality Framework that is composed of four inter-related modules in order to assure the component quality degree. This framework was proposed with basis in a survey on the state-of-the-art in component quality assurance area (Alvaro et al., 2005a), which will be presented on chapter 4. Different from other approaches of the literature (Goulão et al., 2002b), (Beus-Dukic et al., 2003), (Cho et al., 2001), (Gui and Scott, 2007) that provide only isolated aspects to assure the component quality, this thesis tries to investigate

and make available a framework with some necessary mechanisms to execute the component evaluation activities. After that, a set of evaluations are planned to be executed in order to analyze the efficiency of the framework proposed in measuring the quality of the component provided by the main component markets and a Brazilian software factory.

Through these evaluations it is expected a continuously evolution of the whole framework in the way that the software industry can start to trust on it and evaluate its own software components.

1.4 Statement of the Contributions

The main contributions of this research are:

1. A survey of the state-of-the-art of component quality assurance (Alvaro et al., 2005a), in an attempt to analyze this area and possible trends to follow. This survey was developed during the MSc. degree and upgraded during the PhD. degree;
2. The development of a Software Component Quality Framework aiming to provide a well-defined mechanism to evaluate the software component quality (Alvaro et al., 2007a);
3. A refinement of the Component Quality Model (CQM) (Alvaro et al., 2005b), (Alvaro et al., 2005c), (Alvaro et al., 2005d), developed during my MSc. degree (Alvaro, 2005), in order to adapt it to the new standard for Software Product Quality Requirements and Evaluation (SQuaRE) project (ISO/IEC 25000, 2005); and a preliminary evaluation of the CQM also developed during my MSc. degree (Alvaro et al., 2006a), (Alvaro et al., 2006b);
4. The development of a Software Component Evaluation Techniques Model (SCETM) in order to provide different thoroughness levels of evaluation techniques and a set of guidelines for selecting those evaluation levels (Alvaro et al., 2007b);
5. The development of a Component Evaluation Process in order to provide a high quality and consistent evaluation process (Alvaro et al., 2007c), (Alvaro et al., 2007d);

6. The development of a Metrics Framework that is composed of a set of valuable measures to be considered as a starting point during the component evaluations; and
7. The development of an Experimental Study in order to analyze if the proposed framework meets its proposed goals.

1.5 Out of Scope

In order to assure quality to whatever kind of software component it is necessary to develop a well-consistent framework that provide all insights necessary to do this task. However, due to scope limitations, there are other possibilities and work directions that were discarded in this thesis. Thus, the following issues are not directly addressed by this work:

- **Formal Proof:** Meyer (Meyer, 2003) proposes a formal approach in order to acquire trust on software components. His idea is to build software components with fully proved properties. The intention is to develop software components that could be reliable to the software market. This thesis does not consider formal quality assurance mainly because the software component market is not inclined to formally specify their software components. This kind of approach is highly expensive, in terms of development time and level of expertise that is needed, and component developers still do not know if it is cost effective to spend effort in this direction without having specific requirements such as strict time constraints or high reliability. Even so, the Software Component Techniques Model (SCETM), which will be presented in this thesis, provides formal proof evaluation techniques that could be useful in some scenarios, depending on the customer's necessities and the cost/benefit to do so;
- **Prediction of the Component Assembly:** CMU/SEI (Wallnau, 2003) attempts to predict the assembly of software components. This is an ongoing work and the current SEI research line. The SEI's intention is to predict the reliability level of the CBSE in order to determine which quality properties the customer can expect from the system that will be developed using certain components. Besides SEI, there are also other works found in literature that focus on this area

(Stafford et al., 2001), (Hissam et al., 2003). This thesis does not consider this approach for the same reasons that do not consider formal approaches, i.e. it is an expensive approach, and developers are not willing to take high risks in spending effort without knowing if it is cost effective. Also, the first beta tools about this research are available to download since August of 2008 (<http://www.sei.cmu.edu/pacc/starter-kit.html>). One of the SEI's main researchers, Kurt Wallnau (Wallnau, 2004), even states that the viability of this approach is still unknown. Due to this immaturity, this thesis focuses on other aspects of the software component quality.

Additionally, this thesis considered that the first step is evaluate the quality of the component available and after that the quality of the system composed by the selected components. For this reason, the component assembly prediction was not addressed here but should be considered as future work; and

- **Cost Model:** Cost estimation is a key requirement for CBSE before the actual development activities can proceed. Cost is a function of the enterprise itself, its particular development process, the selected solution, and the management and availability of the resource during the development project (Cechich et al., 2003), (Mahmooda et al, 2005). A cost model is useful to help the software engineering during the analysis of the software component available to purchase (or to select or to evaluate). However, it just makes sense when, first, you have defined processes, methods, techniques and tools to execute the selection and/or the evaluation task in order to identify the cost/benefit to purchase or to evaluate a component. In this case, the whole framework that is the basis for component evaluation will be considered in this first moment and, after that a cost model for helping organizations to define if it is viable evaluate certain kind of components (or nor) will be useful.

1.6 Organization of the Thesis

The remainder of this thesis is organized as follows.

Chapter 2 presents a brief overview of the software reuse, software components and component-based development areas. The main concepts of these topics are considered.

Chapter 3 describes the aspects related to the software quality and the software component quality assurance concepts. The intention is to show that software reuse depends on quality.

Chapter 4 presents the survey of the state-of-the-art of the software component quality assurance area that was performed. The failure cases that can be found in literature are also described in this chapter.

Chapter 5 briefly presents the Software Component Quality Framework proposed and its related modules. Moreover, the first module is described, the Component Quality Model (CQM), showing its characteristics, its sub-characteristics, the quality attributes and the metrics that are related to the model.

Chapter 6 presents the Evaluation Techniques Framework which is composed of the Software Component Evaluation Techniques Model (SCETM). The model presents a set of evaluation levels in order to provide flexibility to the component evaluation. Still on, a set of guidance is shown aiming the evaluation team during the levels selection. Moreover, the Metrics Framework and the paradigm adopted to define the metrics are also presented. Some few examples of metrics usage are presented. Additionally, Appendix A is composed of valuable examples of metrics usage in the component evaluation context.

Chapter 7 presents the Software Component Evaluation Process, describing all activities and steps that should be followed to execute the component evaluation activity in a more controllable way.

Chapter 8 presents the definition, planning, operation, analysis and interpretation of the experimental study which evaluates the viability of the proposed process.

Chapter 9 summarizes the main contributions of this work, presents the related works, the concluding remarks and the future work.

Appendix A presents a set of examples of metrics in order to help the software evaluators during the software component evaluation process.

Appendix B presents the template to document the software component evaluation process activity.

Appendix C presents the questionnaires used in the experimental study.

2

Software Reuse

Reuse products, processes and other knowledge will be the key to enable the software industry to achieve a dramatic improvement in productivity and quality that is required to satisfy anticipated growing demands (Basili et al., 1991), (Mohagheghi and Conradi, 2007). However, failing in the adoption of reuse can cost precious time and resources, and may make management skeptical, not willing to try it again. But if your competitors do it successfully and you do not, you may lose a market share and possibly an entire market (Frakes & Isoda, 1994). In other words, if a certain organization does not adopt software reuse before its competitors, it will probably be out of the market, and has a great possibility of crashing.

According to Krueger (Krueger, 1992), software reuse is the process of creating software systems from existing software, instead of building from scratch. Typically, reuse involves the selection, specialization, and integration of artifacts, although different reuse techniques may emphasize some of these aspects. A number of authors (Basili et al., 1996), (Rada et al., 1997), (D'Souza et al., 1999) state that software reuse is the practice of using an artifact in more than one software system.

The most commonly reused software product is source code, which is the final and the most important product of the development activity. In addition to the source code, any intermediary product of the software life cycle can be reused, such as (D'Souza et al., 1999): compiled code, test cases, models, user interfaces and plans/strategies.

According to Basili et al. (Basili et al., 1991), the following assumptions should be considered in the software reuse area:

- **All experience can be reused:** Traditionally, the emphasis has been on reusing concrete assets, mainly source code. This limitation reflects the traditional view that software is equals to code. It ignores the importance of reusing all kinds of software experience, including products, processes, and other knowledge. The term “product” refers to either a concrete document or artifact created during a software project, or a product model describing a class of concrete documents or artifacts with common characteristics. The term “process” refers to either a concrete activity of action aimed at creating some software product, or a process model describing a class of activities or actions with common characteristics. The term “other knowledge” refers to anything useful for software development, including quality and productivity models or models of the application that is being implemented;
- **Reuse typically requires some modification of the assets being reused:** The degree of modification depends on how many, and to what degree, existing assets characteristics differ from those required. The time of modification depends on when the reuse requirements for a project or class of projects are known;
- **Analysis is necessary to determine if, and when, reuse is appropriate:** Reuse pay-off is not always easy to evaluate. There is a need to understand: which are the reuse requirements; how well the available reuse candidates are qualified to meet these requirements; and the mechanisms available to perform the necessary modification;
- **Reuse must be integrated into the specific software development:** Reuse is intended to make software development more effective. In order to achieve this objective, it is needed to tailor reuse practices, methods and tools to the respective development process (decide when, and how, to identify, modify and integrate existing reusable assets.).

Additionally, the primary motivation to reuse software assets is to reduce the time and effort required to build software systems. The quality of software systems is enhanced by reusing quality software assets, which also reduces the time and effort spent in maintenance (Krueger, 1992). Some of the most

important improvements that can be achieved through reuse are summarized below (Lim, 1994).

- **Increased Productivity.** By avoiding redundancy in development efforts, software engineers can accomplish more in less time;
- **Reduced Maintenance Cost.** By reusing high-quality assets, defects are reduced. Furthermore, maintenance efforts are centralized and updates or corrections to one asset can in general propagate easily to consumers;
- **Reduced Training Cost.** By reusing assets, software engineers can easily transfer knowledge that was acquired in different projects. Reusing assets leads to reusing the knowledge associated with these assets. This can greatly reduce the training that is necessary for software engineers to become familiar with the new systems;
- **Increased Quality.** When an asset's cost can be amortized through a large number of usages, it becomes feasible to invest substantial effort in improving its quality. This improvement is seamlessly reflected in the quality of all the products where the asset is used;
- **Support for Rapid Prototyping.** A library of reusable assets can provide a very effective basis for quickly building application prototypes;
- **Specialization.** Reuse allows organizations to capture specialized domain knowledge from producers and leverage this knowledge across the organization.

Sametinger (Sametinger, 1997) agrees with these improvements and presents others that are worth mentioning:

- **Reliability:** Using well-tested assets increases the reliability of a software system. Furthermore, the use of an asset in several systems increases the chance of errors to be detected and strengthens confidence in that asset;
- **Redundant work, development time:** Developing every system from scratch means redundant development of many parts such as user interfaces, communication, basic algorithms, etc. This can be avoided

when these parts are available as reusable assets and can be shared, resulting in less development and less associated time and costs;

- **Time to Market:** The success or failure of a software product is very often determined by its time-to-market. Using reusable assets will result in a reduction of that time; and
- **Documentation:** Even though documentation is very important for the maintenance of a system, it is often neglected. By reusing assets, the amount of documentation that must be written is also reduced. Also, it increases the importance of what is written: only the overall structure of the software system and the newly developed assets have to be documented. The documentation of reusable assets can be shared among many software systems.

Given such important and powerful improvements, the conclusion is that software reuse provides a competitive advantage to the company that adopts it. Some relevant software reuse experience can be found in literature (Endres, 1993), (Griss, 1994), (Frakes & Isoda, 1994), (Joos, 1994), (Griss et al., 1995), (Morisio et al., 2002). Other survey that relates advantages, success and failure cases, myths and inhibitors for software reuse adoption is presented in (Almeida et al., 2007a).

Although the benefits of software reuse are known, it is a complex task to put reuse in practice. Just by grouping some software parts into a library and offering them to the developers is not enough. Instead, the assets have to be carefully designed and developed, in order to offer an effective reuse in all steps of the development process. Besides, there are several factors that directly or indirectly influence the effectiveness of software reuse, such as: well-defined management, software reuse metrics, certification, system repositories, software reuse process, training, organizational incentives, politics, and economical issues, among others.

Some techniques that aim to promote reuse include: domain engineering (Prieto-Díaz, 1990), (Arango, 1994), design patterns (Gamma et al., 1995), product lines (Clements et al., 2001), frameworks (D'Souza et al., 1999), and, component-based development (Brereton et al., 2000), (Meyer et al., 1999).

This last technique is the most commonly used in order to promote reuse and is presented next.

2.1 Component-Based Development (CBD)

Until a few years ago, the development of most software products that are available in the market were based on monolithic blocks, formed by a considerable number of related parts, where these relations were, mostly, implicit. Component-Based Development (CBD) appeared as a new perspective for the software development, aiming at the rupture of these monolithic blocks into interoperable components, decreasing the complexity of the development, as well as its costs, through the use of components that, in principle, are adequate for other applications (Sametinger, 1997).

Only recently, CBD has been considered as an important technique in software development. Some factors fostered new interest in this area, providing necessary motivation to believe that CBD can be now more effective and perform in large scale. Among these factors, some can be mentioned (D'Souza et al., 1999):

- The Development of the Internet, which increases the concerns about distributed computation;
- The change of the systems that are structured in mainframe-based architectures into systems that are based on client/server architectures, leading the developers to consider applications not anymore as monolithic systems but as a set of interoperable subsystems; and
- The use of standards in the applications construction, such as those established by the Object Management Group (OMG) (OMG, 2007), Component Object Model (COM) (Microsoft COM, 2007), CORBA Component Model (CCM) (OMG CCM, 2002) and Enterprise Java Beans (EJB) (DeMichiel, 2002).

According to Vitharana (Vitharana, 2003) the key CBD advantages in future software development efforts are the following:

- **Reduced lead time.** Building complete business applications from an existing pool of components;

- **Leveraged costs developing individual components.** Reusing them in multiple applications;
- **Enhanced quality.** Components are reused and tested in many different applications; and
- **Maintenance of component-based applications.** Easy replacement of obsolete components with new enhanced ones.

The CBD is supposed to reduce the cost and time to market of software applications while increasing their quality. Since components are reused in several occasions, they are likely to be more reliable than software developed from scratch, as they were tested under a larger variety of conditions. Cost and time savings result from the effort that would otherwise be necessary to develop and integrate the functionalities provided by the components in each new software application. In this way, the CBD is the promises of some organizations to promote reuse and the component markets have grown exponentially due to demand for productivity in software development. However, the component area is still immature and future research is needed.

2.1.1 Software Components

The exact concept of component in CBD is not yet a consensus, due to the relative novelty of this area⁴. Each research group characterizes, according to its own context, what a software component is and, thus, there is not a common definition for this term in literature.

Since the first CBD workshop, in 1998, in Kyoto, several definitions have been presented; each one putting into evidence a specific aspect of a component. In Heineman's book (Heineman et al., 2001), a group formed by specialists in CBSE relates that after eighteen months a consensus was achieved about what would be a software component.

⁴ In 1998, the Workshop on Component-Based Software Engineering (CBSE) was first held, in conjunction with the 20th International Conference on Software Engineering (ICSE). Also in 1998, Clemens Szyperski published his first book on software components, which was reedited and the second version launched in 2002 (Szyperski, 2002).

According to these specialists, a software component is a software element that conforms to a component model and that can be independently deployed and composed without modification according to a composition pattern.

Clements Szyperski (Szyperski, 2002) presents a number of definitions of what software components are or should be, trying to define a standard terminology ranging from the semantics of the components and component systems. His concept is sufficient to establish a satisfactory definition about what is a component in CBD, and will be used as basis in this thesis:

*“A software component is a unit of composition with **contractually specified interfaces** and **explicit context dependencies** only. A software component can be **independently deployed** and is subject to **third-party composition**”* (pp. 36).

The *contractually specified interfaces* are important in order to form a common layer between the developer/analyst/architect/designer (i.e. a person who needs a component) and the component developer. The *explicit context dependencies* refer to what the deployment environment must provide so that the components can function properly. Still, for a component to be *independently deployable*, it needs to be well separated from its environment and other components. Finally, for a component to be composed with other *component by a third-party* it must be sufficiently self-contained, i.e. the function that the component performs must be fully performed within itself.

The component interfaces define how this component can be reused and interconnected with other components. The interfaces allow clients of a component, usually other components, to access its services. Normally, a component has multiple interfaces, corresponding to different services.

In (Szyperski, 2002), Szyperski defines an interface as a set of operation signatures that can be invoked by a client. Each operation’s semantics is specified, and this specification plays a dual role as it serves both providers implementing the interface and clients using the interface.

According to Heineman et al. (Heineman et al., 2001), there are two types of interfaces: *provided* and *required*. A component supports a *provided* interface if it contains the implementation of all operations defined by this

interface. On the other hand, a component needs a *required* interface if it depends on other software elements to support this interface.

In these two cases, the connections between the components are accomplished through its interfaces. However, there are cases where this connection is not direct, being necessary the usage of another component, designed exclusively to intermediate this connection. This type of component is usually known as connector (Heineman et al., 2001).

John Williams, in (Heineman et al., 2001), classified software components into three categories:

- **GUI components.** The most prevalent type of component in the marketplace. GUI components encompass all the buttons, sliders, and other widgets used in user interfaces;
- **Service components.** They provide access to common services needed by applications. These include database access, access to messaging and transaction services, and system integration services. One common characteristic of service components is that they all use additional infrastructure or systems to perform their functions; and
- **Domain components.** These are what most developers think of when they talk about business components. Reusable domain components are also difficult to design and build. They may have their own application context dependencies as part of an application infrastructure. Moreover, these components require a high level of domain expertise to build and deploy.

2.2 Summary

This chapter presented the main concepts of software reuse, showing its advantage to the software industries and the main benefits that it can provide when successfully adopted. One of the techniques that promote reuse, component-based development, was also detailed. Additionally, this Chapter presents definitions on software components, and a brief explanation of software components assumptions.

3

Software Quality

The explosive growth of the software industry in recent years has focused attention on the problems long associated with software development: uncontrollable costs, missed schedules, and unpredictable quality. To remain competitive, software factories must deliver high quality products on time and within budget (Slaughter et al., 1998), (Hatton, 2007). The quality in software products was always envisioned by customers.

According to Boegh et al. (Boegh et al., 1993), software quality is something feasible, relative, substantially dynamic and evolutionary, adapting itself to the level of the objectives to be achieved. To reach high quality levels is costly; thus, the important is to focus on the level that is required by the customer.

One of the main objectives of software engineering is to improve the quality of software products, establishing methods and technologies to build software products within given time limits and available resources. Given the direct correlation that exists between software products and processes, the quality area could be divided into two main topics (Pressman, 2005):

- **Software Product Quality:** aiming to assure the quality of the generated product (e.g. ISO/IEC 25000 (ISO/IEC 25000, 2005), ISO/IEC 25051 (ISO/IEC 25051, 2006), ISO/IEC 25040 (ISO/IEC 25040), (McCall et al., 1977), (Boehm et al., 1978), among others); and
- **Software Processes Quality:** looking for the definition, evaluation and improvement of software development processes (e.g. Capability Maturity Model Integrated (CMMI) (CMMI, 2006), ISO/IEC 15504 (ISO/IEC 15504-7, 2008), (Drouin, 1995), among others).

Focusing on software product quality, according to the standard ISO 9000:2000 (ISO 9000, 2005), software quality is the totality of the characteristics of an *entity* that assure itself the capacity of satisfying the *explicit* and *implicit* user's necessities.

It can be noticed that this definition needs complementation, mainly to better define the terms *entity* and *explicit and implicit necessities*. *Entity* is a product/process/service whose quality needs to be assured; the *explicit* necessities are the conditions and objectives captured by the producer; and the *implicit* necessities include the differences between the users, the time evolution, the ethical implications, the security questions, and other subjective visions.

According to the definition, the quality of a product or service is evaluated according to its capability of fulfilling the user necessities. Thus, to guide the quality of a software system means to identify which characteristics need to be developed in order to determine the user necessities and to assure its quality.

However, in general, there is still no consensus about how to define and categorize software product quality characteristics. This thesis follows, as much as possible, a standard terminology, in particular that defined by ISO 9126.

“A quality characteristic is a set of properties of a software product, by which its quality can be described and evaluated. A characteristic may be refined into multiple levels of sub-characteristics.”

This definition suggests that quality is more complex than it appears, i.e., to assure some software quality characteristic, there could be some sub-characteristics. Also, it may be very difficult to determine the quality attributes of each sub-characteristics in order to perform future evaluation and measurement.

An interesting aspect about software quality is that without the customer's recognition, achieving quality is worthless. In this sense, the software must pass through an official certification process, so that the customer may trust that the quality is really present.

Actually, many institutions concern in creating standards to properly evaluate the quality of the software product and software development

processes. In order to provide a general vision, Table 3.1 shows a set of national and international standards in this area.

Table 3.1. Software quality standards.

Standards	Overview
ISO/IEC 9126	Software Products Quality Characteristics
ISO/IEC 14598	Guides to evaluate software product, based on practical usage of the ISO 9156 standard
ISO/IEC 25051	Requirements for Quality of COTS
ISO/IEC 25000	Software Product Quality Requirements and Evaluation
IEEE P1061	Standard for Software Quality Metrics Methodology
ISO 12207	Software Life Cycle Process
NBR ISO 8402	Quality Management and Assurance
NBR ISO 9000-1-2	Model for quality assurance in Design, Development, Test, Installation and Servicing
NBR ISO 90003	Quality Management and Assurance. Application of the ISO 9001 standard to the software development process
ISO/IEC 9000	Quality Management Systems model
CMMI	SEI's model for judging the maturity of the software processes of an organization and for identifying the key practices that are required to increase the maturity of these processes
ISO/IEC 15504	It is a framework for the assessment of software processes
MPS.br	Brazilian software process improvement model

The software market has grown in the last years, as well as the necessity of producing software with quality. Thus, obtaining quality certificates has been a major concern for software companies. Figure 3.1 shows how this tendency influenced the Brazilian software companies until nowadays⁵.

The number of companies looking for standards to assure the quality of their products or processes has grown drastically in the recent past. The graph on the left shows this growth in relation to ISO 9000, which assures the Quality Management and Assurance. The graph on the right shows this growth in relation to CMMI, which assures the software development processes quality. Although this study shows the state of the Brazilian companies, the same tendency can be observed in other countries, as the need for quality assurance in software product and processes is an actual reality of software companies around the world.

⁵ <http://www.softex.br>

ISO 9001

	2002	2003	2004	2005	2006	2007	2008
World	167.124	497.919	660.132	773.867	897.866	979.645	1.105.874
Brazil	1.582	4.012	6.120	8.533	9.014	10.156	12.052

MPS.br

Total per MPS.br level								
Year	A	B	C	D	E	F	G	Total per Year
2005	0	0	0	0	1	3	1	5
2006	2	0	0	1	1	1	7	12
2007	1	0	0	0	1	12	41	55
2005 – 2007	3	0	0	1	3	16	49	72
2008	1	0	0	0	1	9	40	51
2009	0	0	0	0	0	8	5	13
2010	0	0	0	0	0	0	0	0
2008 – 2010	1	0	0	0	1	17	45	64
TOTAL	4	0	0	1	4	33	94	136

CMMi₂₀₀₈

Classification	Country	Nr. CMMi
1	USA	1034
2	China	465
3	India	323
1	UnitedStates	1034
2	China	465
3	India	323
4	Japan	220
5	France	112
6	KoreaRepublic	107
7	Taiwan	88
8	Brazil	79
9	Spain	75
10	UnitedKingdom	71
11	Germany	51
12	Argentina	47
13	Canada	43
14	Malaysia	42
15	Mexico	39
16	Australia	29
17	Egypt	27
18	Chile	20
19	Philippines	20
20	Colombia	18
21	Italy	17
22	Israel	16
23	Singapore	16
24	HongKong	14
25	Pakistan	14

Figure 3.1. Number of companies certified ISO 9000, CMMi and MPS.br

However, there is still no standard or effective process to certificate the quality of pieces of software, such as components. As shown in chapter 1, this is one of the major inhibitors to the adoption of CBD. However, some ideas of software product quality assurance may be seen in the SQuaRE project (described next), which will be adopted as basis for defining a consistent quality framework for software components.

3.1 ISO/IEC 25000 (SQuaRE project)

The SQuaRE (Software Product Quality Requirements and Evaluation) project (ISO/IEC 25000, 2005) has been created specifically to make two standards converge, trying to eliminate the gaps, conflicts, and ambiguities that they present. These two standards are the ISO/IEC 9126 (ISO/IEC 9126, 2001), which define a quality model for software product, and ISO/IEC 14598

(ISO/IEC 14598, 1998), which define a software product evaluation process, based on the ISO/IEC 9126.

Thus, the general objective for this next series is to respond to the evolving needs of users through an improved and unified set of normative documents covering three complementary quality processes: quality requirements specification, measurement and evaluation. The motivation for this effort is to supply those responsible for developing and acquiring software products with quality engineering instruments supporting both the specification and evaluation of quality requirements.

SQuaRE also include criteria for the specification of quality requirements and their evaluation, and recommended measures of software product quality attributes, which can be used by developers, acquirers and evaluators. However, it is important to say that this is an ongoing standard which has been developed/refined since 2005 until now.

SQuaRE consists of 5 divisions as shows in Figure 3.2. The letters ***n*** presented in both divisions represent the possibility to provide more standards in each division.

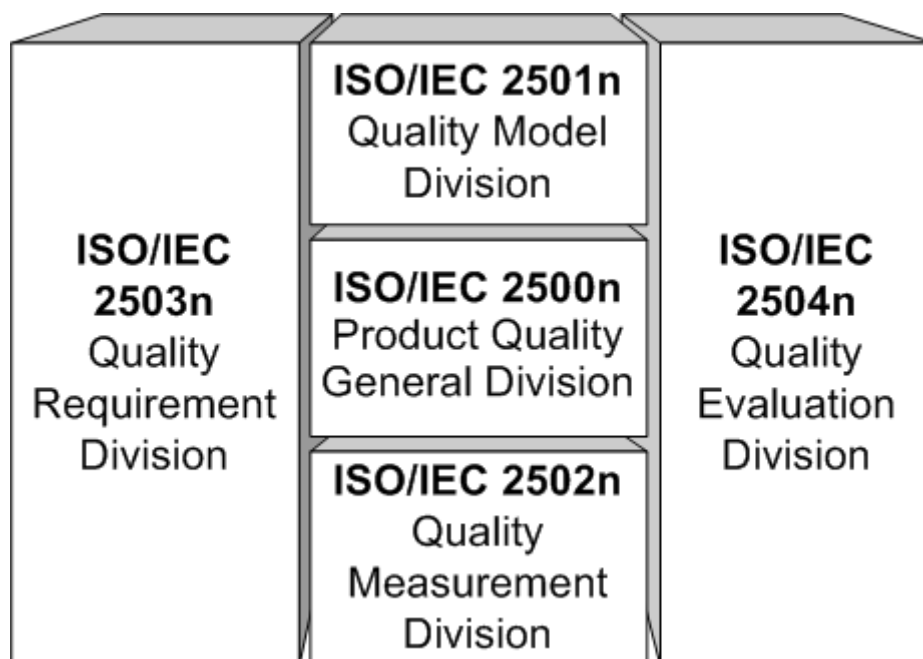


Figure 3.2. SQuaRE Architecture (ISO/IEC 25000, 2005)

The ***Quality Requirements Division (ISO/IEC 2503n)*** (ISO/IEC 25030, 2007) contains the standard for supporting the specification of quality

requirements, either during software product quality requirement elicitation or as an input for an evaluation process. This division includes:

- *Quality requirements and guide*: to enable software product quality to be specified in terms of quality requirements;

The ***Quality Model Division (ISO/IEC 2501n)*** (ISO/IEC 25010) contains the detailed quality model and its specific characteristics and sub-characteristics for internal quality, external quality and quality in use. This division includes:

- *Quality model and guide*: to describe the model for software product internal and external quality, and quality in use. The document presents the characteristics and sub-characteristics for internal and external quality and characteristics for quality in use.

The ***Product Quality General Division (ISO/IEC 2500n)*** (ISO/IEC 25000, 2005) contains an unit standard that defining all common models, terms and definitions referred to by all other standards in the SQuaRE series. Readers are reminded that the Quality Management theme will deal with software products, in contrast to the distinct processes of Quality Management as defined in the ISO 9000 family of standards. This division includes two unit standards:

- *Guide to SQuaRE*: to provide the SQuaRE structure, terminology, document overview, intended users and associated parts of the series, as well as reference models;
- *Planning and management*: to provide the requirements and guidance for planning and management support functions for software product evaluation.

The standards in the ***Quality Measurement Division (ISO/IEC 2502n)*** (ISO/IEC 25020, 2007) were derived from ISO/IEC 9126 and ISO/IEC 14598. This division covers the mathematical definitions and guidance for practical measurements of internal quality, external quality and quality in use. In addition, it includes the definitions for the measurement primitives for all other measures. This module has influence of Goal-Question-Metric (GQM) (Basili et al., 1994), Practical Software Measurement (PSM) (McGarry et al.,

2002) and ISO/IEC 15939 (ISO/IEC 15939, 2007). This theme also contains the Evaluation Module to support the documentation of measurements. This division includes:

- *Measurement reference model and guide*: to present introductory explanations, the reference model and the definitions that is common to measurement primitives, internal measures, external measures and quality in use measures. The document also provides guidance to users for selecting (or developing) and applying appropriate measures;
- *Measurement primitives*: to define a set of base and derived measures, being the measurement constructs for the internal quality, external quality and quality in use measurements;
- *Measures for internal quality*: to define a set of internal measures for quantitatively measuring internal software quality in terms of quality characteristics and sub-characteristics;
- *Measures for external quality*: to define a set of external measures for quantitatively measuring external software quality in terms of quality characteristics and sub-characteristics;
- *Measures for quality in use*: to define a set of measures for measuring quality in use. The document will provide guidance on the use of the quality in use measures.

The ***Quality Evaluation Division (ISO/IEC 2504n)*** (ISO/IEC 25040) contains the standards for providing requirements, recommendations and guidelines for software product evaluation, whether performed by evaluators, acquirers or developers. This division includes:

- *Quality evaluation overview and guide*: to identify the general requirements for specification and evaluation of software quality and to clarify the generic concepts. It will provide a framework for evaluating the quality of a software product and for stating the requirements for methods of software product measurement and evaluation;

- *Process for developers:* to provide requirements and recommendations for the practical implementation of software product evaluation when the evaluation is conducted in parallel with development;
- *Process for acquirers:* to contain requirements, recommendations and guidelines for the systematic measurement, assessment and evaluation of software product quality during acquisition of “commercial-off-the-shelf” (COTS) software products or custom software products, or for modifications to existing software products;
- *Process for evaluators:* to provide requirements and recommendations for the practical implementation of software product evaluation, when several parties need to understand, accept and trust evaluation results;
- *Documentation for the evaluation module:* to define the structure and content of the documentation to be used to describe an Evaluation Module.

The next section will present more details about Quality Model Division, Quality Evaluation Division and Quality Measurement Division. These three divisions are the basis of the SQuaRE project and contain the guidelines/techniques that guide this thesis during the software component quality framework proposal. It is important to say that these five modules of SQuaRE have been in its draft version and, probably, some modification will be done until its final version.

3.2.1 ISO/IEC 2501n (Quality Model Division)

The ISO/IEC 2501n (ISO/IEC 25010) is an evolution of the ISO/IEC 9126 -1 (ISO/IEC 9126, 2001) standard, which provides a quality model for software product. At moment, this division contains only one standard: 25010 – Quality Model and guide. This is an ongoing standard in development.

The *Quality Model Division* does not prescribe specific quality requirements for software, but rather defines a quality model, which can be applied to every kind of software. This is a generic model that can be applied to

any software product by tailoring it to a specific purpose. The ISO/IEC 25010 defines a quality model that comprises six characteristics and 27 sub-characteristics (Table 3.2). The six characteristics are described next:

- **Functionality:** The capability of the software to provide functions which meet stated and implied needs when the software is used under specified conditions;
- **Reliability:** The capability of the software to maintain the level of performance of the system when used under specified conditions;
- **Usability:** The capability of the software to be understood, learned, used and appreciated by the user, when used under specified conditions;
- **Efficiency:** The capability of the software to provide the required performance relative to the amount of resources used, under stated conditions;
- **Maintainability:** The capability of the software to be modified; and
- **Portability:** The capability of software to be transferred from one environment to another.

Table 3.2. Characteristics and Sub-Characteristics in SQuaRE project.

Characteristics	Sub-Characteristics
Functionality	Suitability, Accuracy, Interoperability, Security, Functionality Compliance
Reliability	Maturity, Fault Tolerance, Recoverability, Reliability Compliance
Usability	Understandability, Learnability, Operability, Attractiveness, Usability Compliance
Efficiency	Time Behavior, Resource Utilization, Efficiency Compliance
Maintainability	Analyzability, Changeability, Stability, Testability, Maintainability Compliance
Portability	Adaptability, Installability, Replaceability, Coexistence, Portability Compliance

The usage quality characteristics (i.e. characteristics that can be obtained from the end-user usage feedback) are called *Quality in Use* characteristics and are modeled with four characteristics: effectiveness, productivity, security and satisfaction.

The main drawback of the existing international standards, in this case the ISO/IEC 25010, is that they provide very generic quality models and guidelines, which are very difficult to apply to specific domains such as COTS components and CBSE. Thus, the quality characteristics of this model should be analyzed in order to define the component quality characteristics.

A quality model serves as a basis for determining if a piece of software has a number of quality attributes. In conventional software development, to simply use a quality model is often enough, since the main stakeholders that are interested in software quality are either the developers or the customers that hired these developers. In both cases, the quality attributes may be directly observed and assured by these stakeholders.

3.2.2 ISO/IEC 2504n (Quality Evaluation Division)

The ISO/IEC 2504n (ISO/IEC 25040) is an evolution of the ISO/IEC 14598 (ISO/IEC 14598, 2001) standard, which provides a generic model of an evaluation process, supported by the quality measurements from GQM, PSM and ISO/IEC 15939. This process is specified in four major sets of activities for an evaluation, together with the relevant detailed activities (Figure 3.2). This is an ongoing standard in development.

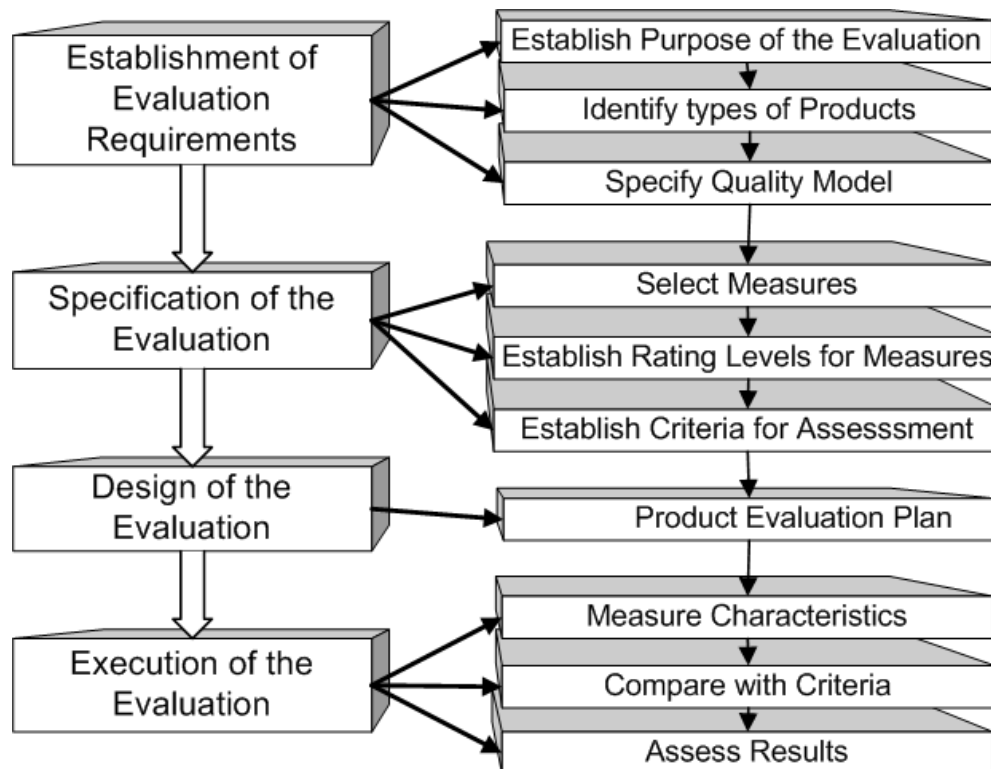


Figure 3.3. ISO/IEC 25040

The ISO/IEC 2504n is divided in five standards: ISO/IEC 25040 – Evaluation reference model and guide; ISO/IEC 25041 – Evaluation modules; ISO/IEC 25042 – Evaluation process for developers; ISO/IEC 25043 – Evaluation process for acquirers; and ISO/IEC 25044 – Evaluation process for evaluators. Besides providing guidance and requirements for the software product evaluation process (ISO/IEC 25040 and ISO/IEC 25041), it provides other three standards that contain guides for different perspectives of software product evaluation: developers, acquires and evaluators.

3.2.3 ISO/IEC 2502n (Quality Measurement Division)

The ISO/IEC 2502n (ISO/IEC 25020, 2007) division tries to improve the quality measurements provided by previous standards like ISO/IEC 9126-2 (external metrics) (ISO/IEC 9126-2, 2003), ISO/IEC 9126-3 (internal metrics) (ISO/IEC 9126-3, 2003) and ISO/IEC 9126-4 (quality in use metrics) (ISO/IEC 9126-4, 2003). However, this standard improves some aspects of quality measurement and the most significantly is the adoption of the Goal-Question-Metrics (GQM) paradigm (Basili et al., 1994).

The ISO/IEC 2502n is divided in five standards: ISO/IEC 25020 - Measurement reference model and guide; ISO/IEC 25021 – Measurement primitives; ISO/IEC 25022 – Measurement of internal quality; ISO/IEC 25023 – Measurement of external quality; and ISO/IEC 25024 – Measurement of quality in use. These standards contain some examples on how to define metrics for different kinds of perspectives, such as internal, external and quality in use.

3.3 Software Component Quality

Once presented the main standards to reach software product quality, this section will discuss the main concepts involving software components quality/certification, which is an attempt to achieve trust in software components.

According to Stafford et al. (Stafford et al., 2001), certification, in general, is the process of verifying a property value associated with something, and providing a certificate to be used as proof of validity.

A “property” can be understood as a discernable feature of “something”, such as latency and measured test coverage, for example. After verifying these properties, a certificate must be provided in order to assure that this “product” has determined characteristics.

Focusing on a certain type of certification, in this case component certification, Councill (Councill, 2001) has given a satisfactory definition about what software component certification is, definition that was adopted in this thesis:

*“Third-party certification is a method to ensure that software components conform to **well-defined standards**; based on this certification, **trusted assemblies** of components can be constructed.”*

To prove that a component conforms to *well-defined standards*, the certification process must provide certificate evidence that it fulfills a given set of requirements. Thus, *trusted assembly* – application development based on third-party composition – may be performed based on the previously established quality levels.

Still, third party certification is often viewed as a good way of bringing trust in software components. Trust is a property of an interaction and is achieved to various degrees through a variety of mechanisms. For example, when purchasing a light bulb, one expects that the base of the bulb will screw into the socket in such a way that it will produce the expected amount of light. The size and threading has been standardized and a consumer “trusts” that the manufacturer of any given light-bulb has checked to make certain that each bulb conforms to that standard within some acceptable tolerance of some set of property values. The interaction between the consumer and the bulb manufacturer involves an implicit trust (Stafford et al., 2001).

In the case of the light-bulb there is little fear that significant damage would result if the bulb did not in fact exhibit the expected property values. This is not the case when purchasing a gas connector. In this case, explosion can occur if the connector does not conform to the standard. Gas connectors are certified to meet a standard, and nobody concerning with safety would use a connector that does not have such a certificate attached. Certification is a mechanism by which trust is gained. Associated with certification is a higher

requirement for and level of trust than can be assumed when using implicit trust mechanisms (Stafford et al., 2001).

When these notions are applied to CBSE, it makes sense to use different mechanisms to achieve trust, depending upon the level of trust that is required.

In order to achieve trust in components, it is necessary to obtain the components that will be evaluated. According to Frakes et al. (Frakes & Terry, 1996), components can be obtained from existing systems through reengineering, designed and built from scratch, or purchased. After that, the components are certified, in order to achieve some trust level, and stored into a repository system, as shows in Figure 3.3.

A component is certifiable if it has properties that can be demonstrated in an objective way, which mean that they should be described in sufficient detail, and with sufficient rigor, to enable their certification (Wallnau, 2003). In order to do that is needed a well-defined component quality model, which incorporates the most common software quality characteristics that are present in the already established models, such as functionally, reliability and performance plus the characteristics that are inherent to CBSE.

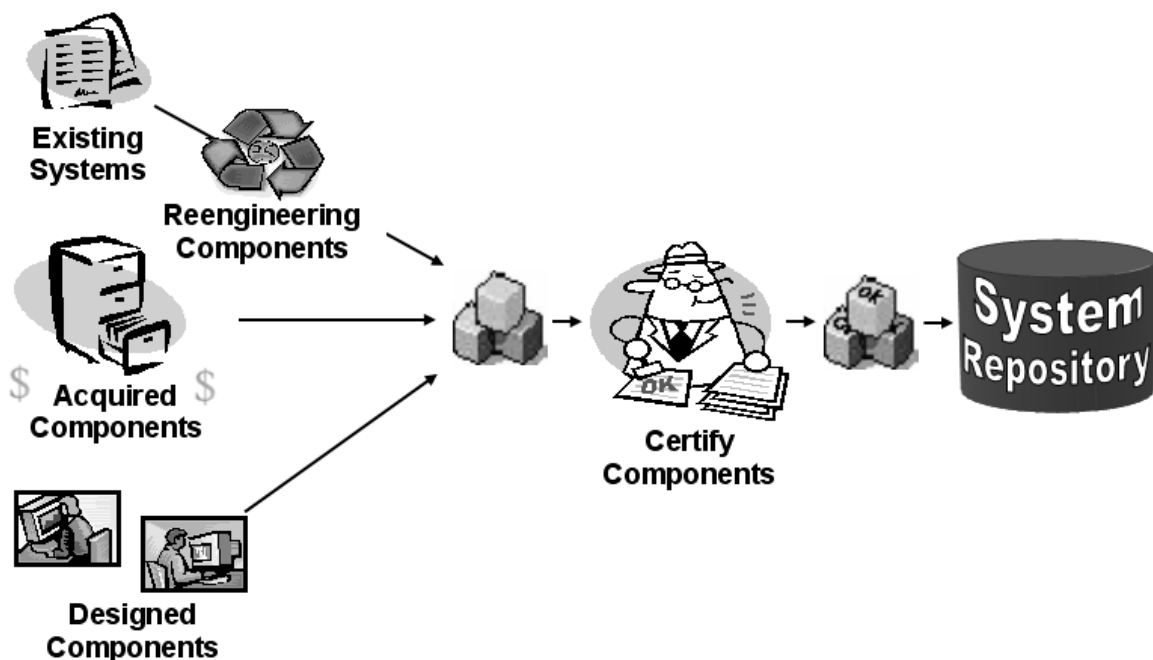


Figure 3.4. Process of obtaining, certifying and storing components

Regarding the certification process, the CBSE community is still far from reaching a consensus on how it should be carried out, what are its requirements

and who should perform it. Still on, third party certification can face some difficulties, particularly due to the relative novelty of this area (Goulao et al., 2002a).

3.4 Summary

This chapter presented the main concepts about software quality and, in the context of this thesis, quality related to software components. It also presented SQuaRE project, a software product quality requirements and evaluation standard that has some ideas regarding component quality assurance. Since trust is a critical issue in CBSE, this chapter also presented some concepts of component certification. As shown, this is a still immature area, and some research is needed in order to acquire the reliability that the market expects from CBSE.

4

Software Component Certification: A Survey

In order to look for plausible answers for the questions discussed in chapter 1, this chapter presents a survey of the state-of-the-art in software component certification research (Alvaro et al., 2005a), in an attempt to analyze this trend in CBSE/CBD and to probe some of the component certification research directions. In this way, works related to certification process in order to evaluate software component quality are surveyed, however the literature contains several works related to software component quality achievement, such as: component testing (Councill, 1999), (Beydeda & Gruhn, 2003), component verification (Wallin, 2002), component contracts (Beugnard et al., 1999), (Reussner, 2003), among others (Kallio et al., 2001), (Cho et al., 2001). Since the focus of this survey is on processes for assuring component quality, it does not cover these works, which deal only with isolated aspects of component quality.

Existing literature is not that rich in reports related to practical software component certification experience, but some relevant research works explore the theory of component certification in academic scenarios. In this sense, this chapter presents a survey of software component certification research, since the early 90's until today. The timeline can be "divided" into two ages: from 1993 to 2001 the focus was mainly on mathematical and test-based models and after 2001 the researches focused on techniques and models based in predicting quality requirements.

During the survey it could be noted that since the beginning the literature presents works related to software component certification which means that the research was trying to propose such a model/standard/insights to certified software components. However, during the years the research started to think

that the market was not mature/prepared for certification in software components. In this way, the works started to proposed ways to evaluate/assure quality in software components, independent of associated any kind of certification.

4.1 Early age: Mathematical and Test-Based Models

Most research published in this period focus on mathematical and test-based models. In 1993, Poore et al. (Poore et al., 1993) developed an approach based on the usage of three mathematical models (sampling, component and certification models), using test cases to report the failures of a system later analyzed in order to achieve a reliability index. Poore et al. were concerned in estimating the reliability of a complete system, and not just the reliability of individual software units, although, they did consider how each component affected the system reliability.

After that, in 1994, Wohlin et al. (Wohlin et al., 1994) presented the first method of component certification using modeling techniques, making it possible not only to certify components but to certify the system containing the components as well. The method is composed of the usage model and the usage profile. The usage model is a structural model of the external view of the components, complemented with a usage profile, which describes the actual probabilities of different events that are added to the model. The failure statistics from the usage test form the input of a certification model, which makes it possible to certify a specific reliability level with a given degree of confidence.

An interesting point of this approach is that the usage and profile models can be reused in subsequent certifications, with some adjustments that may be needed according to each new situation. However, even reusing those models, the considerable amount of effort and time that is needed makes the certification process a hard task.

Two years later, in 1996, Rohde et al. (Rohde et al., 1996) had provided a synopsis of in-progress research and development in reuse and certification of software components at Rome Laboratory of the US Air Force, where a Certification Framework (CF) for software components was being developed.

The purpose of the CF was: to define the elements of the reuse context that are important to certification; to define the underlying models and methods of certification; and, finally, to define a decision-support technique to construct a context-sensitive process for selecting and applying the techniques and tools to certify components. Additionally, Rohde et al. had developed a Cost/Benefit plan that describes a systematic approach to evaluate the costs and benefits of applying certification technology within a reuse program. After analyzing this certification process, Rohde et al. found some points that should be better formulated in order to increase the certification quality, such as the techniques to find errors (i.e. the major errors are more likely to be semantic, not locally visible, rather than syntactic, which this process was looking for) and thus the automatic tools that implements such techniques.

In summary, Rohde et al. considered only the test techniques to obtain the defects result in order to certificate software components. This is only one of the important techniques that should be applied to the component certification.

In 1998, the Trusted Components Initiative (TCI)⁶, a loose affiliation of researchers with a shared heritage in formal interface specification, stood out of the pack representative of TCI is the use of pre/post conditions on APIs (Meyer, 1997), supporting compositional reasoning, but only about a restricted set of behavioral properties of assemblies. Quality attributes, such as security, performance, availability, and so forth, are beyond the reach of these assertion languages.

The major advanced achievement of TCI was the practical nature of the experiments conducted.

In this same year, Voas (Voas, 1998) defined a certification methodology using automated technologies, such as black-box testing and fault injection to determine if a component fits into a specific scenario.

This methodology uses three quality assessment techniques to determine the suitability of a candidate COTS (Commercial Off-The-Shelf) component: **(i) Black-box component testing** is used to determine whether the component quality is high enough; **(ii) System-level fault injection** is used to determine

⁶ <http://www.trusted-components.org>

how well a system will tolerate a faulty component; **(iii) Operational system testing** is used to determine how well the system will tolerate a properly functioning component, since even these components can create system wide problems.

The methodology can help developers to decide whether a component is right for their system or not, showing how much of someone else's mistakes the components can tolerate.

According to Voas, this approach is not foolproof and perhaps not well-suited to all situations. For example, the methodology does not certify that a component can be used in all systems. In other words, Voas focused his approach in certifying a certain component within a specific system and environment, performing several types of tests according to the three techniques that were cited before.

Another work involving component test may be seen in (Wohlin and Regnell, 1998), where Wohlin and Regnell extended their previous research (Wohlin et al., 1994), now, focusing on techniques for certifying both components and systems. Thus, the certification process includes two major activities: **(i)** usage specification (consisting of a usage model and profiles) and **(ii)** certification procedure, using a reliability model.

The main contribution of that work is the division of components into classes for certification and the identification of three different ways for certifying software systems: **(i) Certification process**, in which the functional requirements implemented in the component are validated during usage-based testing in the same way as in any other testing technique; **(ii) Reliability certification of component and systems**, in which the component models that were built are revised and integrated to certificate the system that they form; and, **(iii) Certify or derive system reliability**, where the focus is on reusing the models that were built to certify new components or systems.

In this way, Wohlin and Regnell provided some methods and guidelines for suitable directions to support software component certification. However, the proposed methods are theoretical without experimental study. According to Wohlin et al., *“both experiments in a laboratory environment and industrial case studies are needed to facilitate the understanding of component*

reliability, its relationship to system reliability and to validate the methods that were used only in laboratory case studies” (pp. 09). Until now, no progress in those directions was achieved.

The state of the art, up to around 1999, was that components were being evaluated only with the results of the tests performed in the components. However, there was no well-defined way to measure the efficiency of the results. In 2000, Voas et al. (Voas et al., 2000) defined some dependability metrics in order to measure the reliability of the components, and proposed a methodology for systematically increasing dependability scores by performing additional test activities. This methodology helps to provide better quality offerings, by forcing the tests to only improve their score if the test cases have a greater tendency to reveal software faults. Thus, these metrics and methodology do not consider only the number of tests that a component received but also the “fault revealing” ability of those test cases. This model estimates the number of test cases necessary in order to reveal the seeded errors. Beyond this interesting point, the Voas et al. work was applied to a small amount of components into an academic scenario. Even so, the methodology presented some limitations, such as: the result of the “fault revealing” ability was not satisfactory; the metrics needed more precision; and, there was a lack of tools to automate the process. Additionally, this methodology was not applied to the industry, which makes its evaluation difficult.

In 2001, Morris et al. (Morris et al., 2001) proposed an entirely different model for software component certification. The model was based on the tests that developers supply in a standard portable form. So, the purchasers can determine the quality and suitability of purchased software.

This model is divided in four steps: **(i) Test Specification**, which uses XML (eXtensible Markup Language) files to define some structured elements that represent the test specifications; **(ii) Specification Document Format**, which describes how the document can be used or specified by a tester; **(iii) Specified Results**, which are directly derived from a component’s specification. These results can contain an exact value or a method for computing the value, and are stored in the test specifications of the XML elements; and, **(iv) Verifier**, which evaluates a component. In other words,

Morris built a tool that reads these XML files and performs the respective tests in the components, according to the parameters defined in XML files.

This model has some limitations for software component certification, such as: additional cost for generating the tests, developer resources to build these tests, and the fact that it was conceived only for syntactic errors. However, as cited above, the majority of errors are likely to be semantic, not locally visible, rather than syntactic, which was the aim of the model.

Although this period was mainly focused on mathematical and test-based models, there were different ideas around as well. A **first work** that can be cited was published in 1994. Merrit (Merrit, 1994) presented an interesting suggestion: the use of components certification levels. These levels depend on the nature, frequency, reuse and importance of the component in a particular context, as it follows:

- **Level 1:** A component is described with keywords and a summary is stored for automatic search. No tests are performed; the degree of completeness is unknown;
- **Level 2:** A source code component must be compiled and metrics are determined;
- **Level 3:** Testing, test data, and test results are added; and
- **Level 4:** A reuse manual is added.

Although simple, these levels represent an initial component quality model. To reach the next level, the component efficiency and documentation should be improved. The closer to level four, the higher is the probability that the component is trustworthy and may be easily reused. Moreover, Merrit begins to consider other important characteristics related to component certification, such as attaching some additional information to components, in order to facilitate their recovery, defining metrics to assure the quality of the components, and providing a component reutilization manual in order to help its reuse in other contexts. However, this is just a suggestion of certification levels and no practical work was actually done to evaluate it.

A **second work** that goes beyond mathematical and test-based models, discussing important issues of certification, was a panel presented in ICSE'2000

- International Conference on Software Engineering, by Council et al. (Council et al., 2000). The panel had the objective of discussing the necessity of trust assurance in components. CBSE researchers participated in this discussion, and all of them agreed that the certification is essential to increase software component adoption and thus its market. Through certification, consumers may know the trust level of components before acquiring them.

Besides these contributions, the main advance achieved in this period was the fact that component certification began to attract attention and started to be discussed in the main CBSE workshops (Crnkovic et al., 2001), (Crnkovic et al., 2002).

4.2 Second age: Testing is not enough to assure component quality

After a long time considering only tests to assure component reliability levels, around 2000, the research on the area started to change focus, and other issues began to be considered in component certification, such as reuse level degree, reliability degree, among other properties.

In 2001, Stafford et al. (Stafford et al., 2001) developed a model for the component marketplaces that supports prediction of system properties prior to component selection. The model is concerned with the question of verifying functional and quality-related values associated with a component. This work introduced notable changes in this area, since it presents a CBD process with support for component certification according to the credentials, provided by the component developer. Such credentials are associated to arbitrary properties and property values with components, using a specific notation such as $\langle \text{property}, \text{value}, \text{credibility} \rangle$. Through credentials, the developer chooses the best components to use in the application development based on the “credibility” level.

Stafford et al. also introduced the notion of *active component dossier*, in which the component developer packs a component along with everything needed for the component to be used in an assembly. A *dossier* is an abstract component that defines certain credentials, and provides benchmarking mechanisms that, given a component, will fill in the values of these credentials.

Stafford et al. finalized their work with some open questions, such as: how to certify measurement techniques? What level of trust is required under different circumstances? Are there other mechanisms that might be used to support trust? If so, are there different levels of trust associated with them and can knowledge of these differences be used to direct the usage of different mechanisms under different conditions?

Besides these questions, there are others that must be answered before a component certification process is achieved, some of these apparently as simple as: what does it mean to trust a component? (Hissam et al., 2003), or as complex as: what characteristics of a component make it certifiable, and what kinds of component properties can be certified? (Wallnau, 2003).

Concurrently, in 2001, Councill (Councill, 2001) had examined other aspects of component certification, describing, primarily, the human, social, industrial, and business issues required to assure trusted components. These issues were mainly concerned with questions related to software faults and in which cases these can be prejudicial to people; the cost-benefit of software component certification; the certification advantage to minimize project failures, and the certification costs related with the quantity of money that the companies will save with this technique. The aspects considered in this work had lead Councill in assuring, as well as Heineman (Heineman et al., 2000), (Heineman et al., 2001), Crnkovic (Crnkovic, 2001) and Wallnau (Wallnau, 2003), that certification is strictly essential for software components.

In this same year, Woodman et al. (Woodman et al., 2001) analyzed some processes involved in various approaches to CBD and examined eleven potential CBD quality attributes. According to Woodman et al., only six requirements are applicable to component certification: *Accuracy*, *Clarity*, *Replaceability*, *Interoperability*, *Performance* and *Reliability*. But these are “macro-requirements” that must be split into some “micro-requirements” in order to aid in the measurement task. Such basic requirement definition is among the first efforts to specify a set of properties that should be considered when dealing with component certification. However, all of these requirements should be considered and classified in an effective component quality model in order to achieve a well-defined certification process.

In 2002, Comella-Dorda et al. (Comella-Dorda et al., 2002) proposed a COTS software product evaluation process. The process contains four activities, as follows: **(i) Planning the evaluation**, where the evaluation team is defined, the stakeholders are identified, the required resources are estimated and the basic characteristics of the evaluation activity are determined; **(ii) Establishing the criteria**, where the evaluation requirements are identified and the evaluation criteria are constructed; **(iii) Collecting the data**, where the component data are collected, the evaluation plan is done and the evaluation is executed; and **(iv) Analyzing the data**, where the results of the evaluation are analyzed and some recommendations are given.

With the same objective, in 2003 Beus-Dukic et al. (Beus-Dukic et al., 2003) proposed a method to measure quality characteristics of COTS components, based on the latest international standards for software product quality (ISO/IEC 9126, ISO/IEC 12119 and ISO/IEC 14598). The method is composed of four steps, as follows: **(i) Establish evaluation requirements**, which include specifying the purpose and scope of the evaluation, and specifying evaluation requirements; **(ii) Specify the evaluation**, which include selecting the metrics and the evaluation methods; **(iii) Design the evaluation**, which considers the component documentation, development tools, evaluation costs and expertise required in order to make the evaluation plan; and **(iv) Execute the evaluation**, which include the execution of the evaluation methods and the analysis of the results.

Although similar to the previous work Comella-Dorda et al. and Beus-Dukic et al.'s work are based on international standards for software product quality, basically, the ISO 14598 principles. However, the method proposed was not evaluated in a real case study, and, thus its real efficiency in evaluating software components is still unknown.

In 2003, Hissam et al. (Hissam et al., 2003) introduced Prediction-Enabled Component Technology (PECT) as a means of packaging predictable assembly as a deployable product. PECT is meant to be the integration of a given component technology with one or more analysis technologies that support the prediction of assembly properties and the identification of the required component properties and their possible certifiable descriptions. This

work, which is an evolution of Stafford et al.'s work (Stafford et al., 2001), attempts to validate the PECT and its components, giving credibility to the model, which will be further discussed in this section.

Another approach was proposed by McGregor et al. in 2003 (McGregor et al., 2003), defining a technique to provide component-level information to support prediction of assembly reliabilities based on properties of the components that form the assembly. The contribution of this research is a method for measuring and communicating the reliability of a component in a way that it becomes useful to describe components intended to be used by other parties. The method provides a technique for decomposing the specification of the component into logical pieces about which it is possible to reason.

In McGregor et al.'s (McGregor et al., 2003), some “roles” (component services) are identified through the component documentation and the developer may have listed the roles, identifying the services that participate in those roles. The reliability test plan identifies each of the roles and, for each role, the services that implement the role, providing reliability information about each role that the component is intended to support. However, this method is not mature enough in order to have its real efficiency and efficacy evaluated in a proper way. According to McGregor et al., this method is a fundamental element in an effort to construct a PECT (Hissam et al., 2003).

During 2003, a CMU/SEI's report (Wallnau, 2003) extended Hissam et al. work (Hissam et al., 2003), describing how component technology can be extended in order to achieve Predictable Assembly from Certifiable Components (PACC). This new initiative is developing technology and methods that will enable software engineers to predict the runtime behavior of assemblies of software components from the properties of those components. This requires that the properties of the components are rigorously defined, trusted and amenable to certification by independent third parties.

SEI's approach to PACC is PECT, which follows Hissam et al.'s work (Hissam et al., 2003). PECT is still an ongoing research project that focuses on analysis – in principle any analysis could be incorporated. It is an abstract model of a component technology, consisting of a construction framework and a reasoning framework. In order to concretize a PECT, it is necessary to choose an

underlying component technology, to define restrictions on that technology to allow predictions, and to find and implement proper analysis theories. The PECT concept is portable, since it does not include parts that are bound to any specific platform. Figure 4.1 shows an overview of this model.

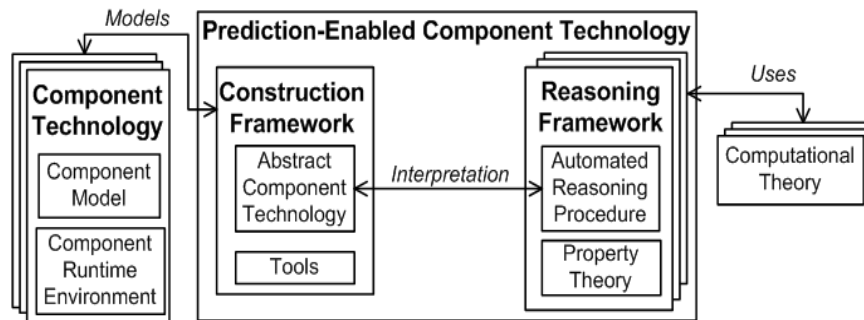


Figure 4.1. Structure of Prediction-Enabled Component Technology (Wallnau, 2003).

A system built within the PECT framework can be difficult to understand, due to the difficulty of mapping the abstract component model into the concrete component technology. It is even possible that systems that look identical at the PECT level behave differently when realized on different component technologies.

Although PECT is highly analyzable and portable, it is not very understandable. In order to understand the model, the mapping to the underlying component technology must be understood as well.

This is the current SEI research framework for software component quality. This model requires a better maturation by the software engineering community in order to achieve trust on it. Therefore, some future works are being accomplished, such as: tools development to automate the process, the applicability analysis of one or more property theories, non-functional requirements certification, among others. Moreover, there is still the need for applying this model in industry scenarios and evaluating the validity of the certification.

In another work, in 2003, Meyer (Meyer, 2003) highlighted the main concepts behind a trusted component along two complementary directions: a “low road”, leading to certification of existing components (e.g. defining a component quality model), and a “high road”, aimed at the production of

components with fully proved correctness properties. In the first direction, Meyer was concerned with establishing the main requirements that a component must have. Meyer's intention is to define a component quality model, in order to provide a certification service for existing components – COM, EJB, .NET, OO libraries. This model - still under development - has five categories. When all properties in one category are achieved, the component has the corresponding quality level.

In the second direction, Meyer analyzed the previous work in order to construct a model that complements its certification process. The intention is to develop components with mathematically proved properties.

In 2003, Gao et al. (Gao et al., 2003) published the first book about software component quality, called “*Testing and Quality Assurance for Component Based Software*”. The book presented the state-of-the-art in component-based software testing, showing the current issues, challenges, needs, and solutions in this critical area. It also discusses the advances in component-based testing and quality assurance.

In 2005, Alvaro et al. (Alvaro et al., 2005c) presented a Component Quality Model describing mainly the quality attributes and related metrics for the components evaluation. The model developed was based on ISO/IEC 9126 and a set of updates in the Characteristics and Sub-Characteristics were provided in order to be used in a software component context. At least, some metrics were presented in order to provide means to measure the quality characteristic proposed in the model.

Later, Alvaro et al. (Alvaro et al., 2006a) presented a preliminary evaluation of the Component Quality Model presented in (Alvaro et al., 2005c) in order to analyze the results of using the model. In that way, the results were considered satisfactory once five from six null hypotheses were rejected during the experimental study.

In 2007, Andreou & Tziakouris (Andreou & Tziakouris, 2007) proposed a quality framework for developing and evaluating original components, along with an application methodology that facilitates their evaluation. The framework was based on the ISO/IEC 9126 quality model which is modified and refined in order to reflect better the notion of original components. The quality

model introduced can be tailored according to the organization and the domain needs of the targeted component. The idea of this model is the same of the model proposed in (Alvaro, 2005) however some quality characteristics of ISO/IEC 9126 were eliminated from the model.

At least, in 2008, (Choi et al., 2008) proposed an in-house Component Quality Model which includes metrics for component quality evaluation, tailoring guidelines for evaluations, and reporting formats of evaluations. The model proposed was based on ISO/IEC 9126 and Choi et al. have applied this Component Quality Model to embedded system development projects. The future works will try to automate some quality characteristics through a set of tools developed in Samsung – Korea labs. The model proposed in this work is specific to embedded system domain which means that the literature started to tailor some models to specific kind of domains.

4.3 Failures in Software Component Certification

The previous section presented a survey related to the component certification research. This section describes two failure cases that can be found in the literature. The **first** failure occurred in the US government, when trying to establish criteria for certifying components, and the **second** failure happened with an IEEE committee, in an attempt to obtain a component certification standard.

(i) Failure in National Information Assurance Partnership (NIAP). One of the first initiatives attempting to achieve trusted components was the NIAP. The NIAP is an U.S. Government initiative originated to meet the security testing needs of both information technology (IT) consumers and producers. NIAP is a collaboration between the National Institute of Standards and Technology (NIST) and the National Security Agency (NSA). It combines the extensive IT security experience of both agencies to promote the development of technically sound security requirements for IT products, systems and measurement techniques.

Thus, from 1993 until 1996, NSA and the NIST used the Trusted Computer Security Evaluation Criteria (TCSEC), a.k.a. “Orange Book.”⁷, as the basis for the

⁷ <http://www.radium.ncsc.mil/tpep/library/tcsec/index.html>

Common Criteria⁸, aimed at certifying security features of components. Their effort was not crowned with success, at least partially because it had defined no means of composing criteria (features) across classes of components and the support for compositional reasoning, but only for a restricted set of behavioral assembly properties (Hissam et al., 2003).

(ii) Failure in IEEE. In 1997, a committee was gathered to work on the development of a proposal for an IEEE standard on software components quality. The initiative was eventually suspended, in this same year, since the committee came to a consensus that they were still far from getting to the point where the document would be a strong candidate for a standard (Goulao et al., 2002a).

4.4 Conclusion of the Study

Figure 4.2 summarizes the timeline of research on the software component certification area, where the dotted line marks the main change in this research area, from 1993 to 2008 (Figure 4.2). Besides, there were two projects that failed (represented by an “X”), one project that was too innovative for its time (represented by a circle) and two projects related to certification concepts, the requirements and discussion about how to achieve component certification (represented by a square). The arrows indicate that a work was extended by another.

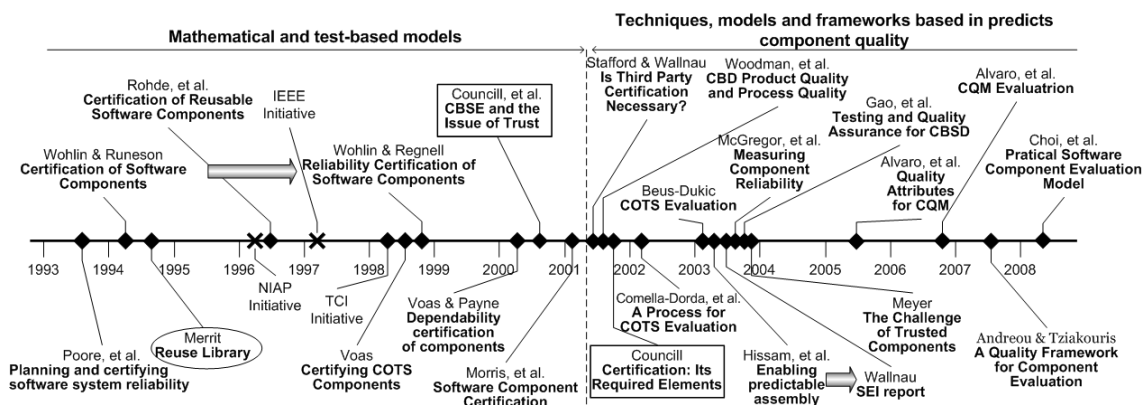


Figure 4.2. Research on software component certification timeline.

The research in the component certification area follows two main directions based on: **(i) Formalism:** How to develop a formal way to predict

⁸ <http://csrc.nist.gov/cc>

component properties? (e.g. PECT model) and How to build components with fully proved correctness properties? (e.g. Meyer’s “high road” model); and **(ii) Component Quality Model:** How to establish a well-defined component quality model and what kinds of component properties can be certified? (e.g. Meyer’s “low road” model).

However, these works still need some effort to conclude the proposed models and to prove its trust, and require a definition on which requirements are essential to measure quality in components. Even so, a unified and prioritized set of CBSE requirements for reliable components is a challenge in itself (Schmidt, 2003).

4.5 Summary

This chapter presented a survey related to the state-of-the-art in the software component certification research. Some approaches found in the literature, including the failure cases, were described. Through this survey, it can be noticed that software components certification is still immature and further research is needed in order to develop processes, methods, techniques, and tools aiming to obtain well-defined standards for component quality evaluation.



Software Component Quality Framework and Component Quality Model

After the survey of the state-of-the-art in software component certification research was accomplished (presented in chapter 4), it was noted that there is a lack of processes, methods, techniques and tools available on the literature to be used for evaluating component quality. This need for processes, methods, techniques and tools to perform the component quality assurance is pointed out by several researchers (Voas, 1998), (Morris et al., 2001), (Wallnau, 2003), (Alvaro et al., 2005), and was evidenced by studies accomplished by SEI (Bass et al., 2003), Softex (Softex, 2007) and Lucrédio et al. (Lucrédio et al., 2007). Most researchers agree that component quality is an essential aspect of the CBSE adoption and software reuse success.

Motivated by the needs pointed out in chapter 4, a software component quality framework is proposed. The framework tries to be as complete as possible, in order to provide insights required for the evaluator to execute the component evaluation. Its idea is to improve the lack of consistency between the available standards for software product quality (ISO/IEC 9126), (ISO/IEC 14598), (ISO/IEC 12119), also including the software component quality context. These standards provide a high-level definition of characteristics and metrics for software product but don't provide ways to be used in an effective way, becoming very difficult to apply them without acquiring more knowledge from other sources (i.e. books, papers, etc.).

Thus, the main goal of the proposed component quality framework is to provide modules that are consistent enough, each one complementing each other, in order to be self-sufficiency (i.e. the information needed to do the

component evaluation task are available). In this task, a recent standard is useful, the SQuaRE project⁹, which has been developed/improved until nowadays. In this way, based on this standard, it is necessary to define a Component Quality Model (CQM)¹⁰. However, there are several difficulties in the development of such a model, such as: **(1)** which quality characteristics should be considered, **(2)** how to evaluate them, and **(3)** who should be responsible for such evaluation (Goulão et al., 2002a).

In general, one of the core goals to achieve quality in components is to acquire reliability on it and, in this way, increase the component market adoption. Usually, the software component evaluation occurs through models that measure its quality. These models describe and organize the component quality characteristics that will be considered during the evaluation. So, to measure the quality of a software component it is necessary to develop a quality model.

In this way, an overview of the proposed framework is presented in this chapter, and it also presents a Component Quality Model, its characteristics and sub-characteristics, and the quality attributes that compose the model. Next chapters will present the others modules with more details.

5.1 Overview of the Framework

Based on a robust framework for software reuse (Almeida et al., 2004) – presented on chapter 1 – which is being developed by the Reuse in Software Engineering (RiSE) group, there must be a layer that considers the quality aspects of the assets developed during the software reuse process. This layer is essential once the assets reused without quality can decrease the improvements expected with software reuse benefits (Frakes & Fox, 1995) i.e. reusable assets

⁹ The SQuaRE project presented in Chapter 3 has been developed with this intention but this initiative started in 2005 and until nowadays there are huge efforts around the world in order to finish the first version of the whole standard.

¹⁰ The model proposed here is an evolution of the Component Quality Model presented on Alvaro's MSc. Dissertation (Alvaro, 2005), where the previous model was based on ISO/IEC 9126 and this presented here has becoming compatible to the SQuaRE project (ISO/IEC 25000, 2005).

without quality can impact the quality of the whole system and some effort will be needed to correct the errors and faults found. According to Councill (Councill, 2001), it is better to develop your components and system from scratch than reuse an asset without quality or with unknown quality, instead of having the risk to impact the project planning, quality and time-to-market.

However, the process of evaluating software component quality is not a simple one. First, there should be a **component quality model**. Differently from other software product quality models, such as (McCall et al., 1977), (Boehm et al., 1978), (Hyatt et al., 1996), (ISO/IEC 9126, 2001), (Georgiadou, 2003), this model should consider Component-Based Development (CBD) characteristics, sub-characteristics and describe quality attributes that are specific to promote reuse. Moreover, it should be consistent enough to provide characteristics that complement each other with the intention of providing a good quality model to the software component context.

With a component quality model in hand, there must be a series of **evaluation techniques** that allow one to evaluate if a component conforms to the model. It is very useful to provide the techniques that can be correlated to the quality characteristics proposed on the quality model. Thus, this module is very important once it is impossible to evaluate the quality of the software component without a set of efficient evaluation techniques that cover one or more quality characteristics provided by the component quality model (first module). Moreover, the techniques provided should consider a set of levels in order to provide different depth of evaluation.

Consistent and good evaluation results can only be achieved by following a high quality and consistent evaluation process (Comella-Dorda et al., 2002). So, the correct usage of these evaluation techniques should follow a well-defined and controllable **component evaluation process**. Through this process the evaluation can be carried out more precisely and effectively, once the evaluator has some guidelines, templates and activities to follow. In addition, the main goal of a well-defined process is that the certification can be repeatable and reproducible among different evaluators.

Finally, a set of **metrics** are needed, in order to track the components properties, the completeness of the component quality model proposed, the efficiency of the evaluation techniques used and the enactment of the certification process. The metrics are important to obtain the feedback of the whole framework and improve the quality of the modules.

These four main issues **(i)** a Component Quality Model, **(ii)** a Evaluation Techniques Framework, **(iii)** a Metrics Framework, and **(iv)** a Evaluation Process, are the modules of a Software Component Quality Framework (Figure 5.1).

The framework will allow that the components produced in a *Software Reuse Environment* are certified before being stored in a *Repository System*. In this way, software engineers would have a greater degree of trust in the components that are being reused and becoming more encouraged reusing assets from the repository system.

According to the three perspectives considered in the SQuaRE project, which was presented on chapter 3, this framework could be used according to the following perspectives: *acquirers*, *evaluators* and *developers*. In the **first** perspective, it should be considered if the customer has a set of components that contain the same requirements and functionalities, but have different costs, performance, and reliability attributes, among others. In this case, the framework could be applied in order to define which component best fits the customer needs and application/domain context. The **second** perspective should be considered for software component evaluation required by companies in order to achieve trust on its own components, looking for developing more reliable applications or to sell components with higher quality. The **third** perspective should be considered for developers know the way that the framework will evaluate its components and to develop the component according to the framework, looking for increasing the component quality.

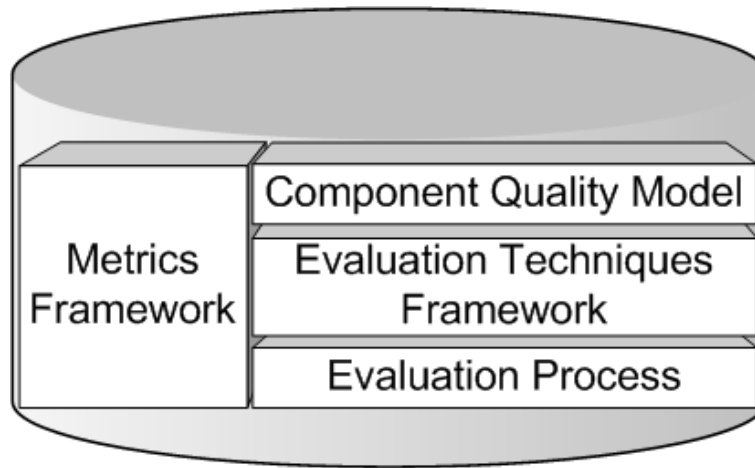


Figure 5.1. Software Component Quality Framework.

5.2 The Component Quality Model (CQM)

The CQM proposed is based on SQuaRE project (ISO/IEC 25000, 2005), named ISO/IEC 25010 standard, with adaptations for components. The model is composed of marketing characteristics and some relevant component information that is not supported in other component quality models (Goulão et al., 2002b), (Bertoa et al., 2002), (Meyer, 2003), (Simão et al., 2003), which will be presented next.

Although recent, some component quality models (Goulão et al., 2002b), (Bertoa et al., 2002), (Meyer, 2003), (Simão et al., 2003) are described in the literature and analyzed in order to identify directions for proposing a well-defined quality model for software component evaluation. The negative and positive points of each model were considered in this study, aiming the identification of the characteristics that are really important to such a model.

In this way, after analyzing these models and the ISO/IEC 25010, a CQM was developed¹¹ (Alvaro et al., 2005b), (Alvaro et al., 2005c), (Alvaro et al., 2005d). The proposed CQM is composed of seven characteristics, as follows:

- **Functionality:** This characteristic expresses the ability of a software component to provide the required services and functions, when used under specified conditions;

¹¹ The CQM was proposed during my Master degree in Computer Science (Alvaro, 2005) and during this time it was made compatible with the evolution of the ISO/IEC 9126 standard, the SQuaRE project.

- **Reliability:** This characteristic expresses the ability of the software component to maintain a specified level of performance, when used under specified conditions;
- **Usability:** This characteristic expresses the ability of a software component to be understood, learned, used, configured, and executed, when used under specified conditions;
- **Efficiency:** This characteristic expresses the ability of a software component to provide appropriate performance, relative to the amount of resources used;
- **Maintainability:** This characteristic describes the ability of a software component to be modified;
- **Portability:** This characteristic is defined as the ability of a software component to be transferred from one environment to another; and
- **Marketability:** This characteristic expresses the marketing characteristics of a software component.

Although the model is proposed following the ISO/IEC 25010 standard, some changes were made in order to develop a consistent model to evaluate software components:

- The characteristics that were identified as relevant to the component context were maintained;
- One sub-characteristic that proved to be not interesting to evaluate components was eliminated (Analyzability);
- The name of one of the sub-characteristics was changed in order to adequate it to the component context (from Instalability to Deployability);
- Another level of characteristics was added, containing relevant marketing information for a software component evaluation process;
- Some sub-characteristics that complement the CQM with important component information were established.

5.2.1 Changes in relation to ISO/IEC 25010

According to ISO/IEC 25010 (ISO/IEC 25010, 2005), the model should be tailored in order to represent better the domain/context that will be indented to

work during the evaluation. In this sense, table 5.1 summarizes the changes that were performed in relation to ISO/IEC 25010. The characteristics and sub-characteristics that are represented in bold were not present in ISO/IEC 25010. They were added due to the need for evaluating certain CBSD-related properties that were not covered on ISO/IEC 25010. The sub-characteristic that is crossed was present in ISO/IEC 25010, but was removed from the proposed model. Finally, the sub-characteristic in italics had its name changed.

Table 5.1. Changes in the Proposed Component Quality Model, in relation to ISO/IEC 25010.

Characteristics	Sub-Characteristics
Functionality	Suitability Accuracy Interoperability Security Self-sufficiency Compliance
Reliability	Maturity Recoverability Fault Tolerance Compliance
Usability	Understandability Configurability Learnability Operability Compliance
Efficiency	Time Behavior Resource behavior Scalability Compliance
Maintainability	Analyzability Stability Changeability Testability Compliance
Portability	<i>Deployability</i> Replaceability Adaptability Reusability Compliance
Marketability	Price Time to market Targeted market Licensing

The ***Self-sufficiency*** sub-characteristic is intrinsic to software components and must be analyzed.

The ***Configurability*** is an essential feature that the developer must analyze in order to determine if a component can be easily configured. Through this sub-characteristic, the developer is able to preview the complexity of deploying the component into a certain context.

The ***Scalability*** sub-characteristic is relevant to the model because it expresses the ability of the component to support major data volumes processing. So, the developer will know if the component supports the data demand of his/her application.

The reason why software factories have adopted component-based approaches to software development is the promise of reuse. Thus, the ***Reusability*** sub-characteristic is very important to be considered in this model.

A brief description of each new sub-characteristic is presented, as follows:

- **Self-sufficiency:** The function that the component performs must be fully performed within itself;
- **Configurability:** The ability of the component to be configurable (e.g. through a XML file or a text file, the number of parameters, etc.);
- **Scalability:** The ability of the component to accommodate major data volumes without changing its implementation; and
- **Reusability:** The ability of the component to be reused. This characteristic evaluates the reusability level through some points, such as: the abstraction level, if it is platform-specific or not, if the business rules are interlaced with interface code or *SQL* code, among others.

Additionally, one sub-characteristic was removed in order to adequate the model to the component context (crossed). In the ***Maintainability*** characteristic, the ***Analyzability*** sub-characteristic disappeared. Its main concern, according to ISO/IEC 25010, is to assert if there are methods for performing auto-analysis, or identifying parts to be modified. Since a component is developed with some functionality in mind, this kind of auto-

analysis methods is rarely developed. In fact, practical experience has shown that components do not have *Analyzability* characteristics (Bertoa et al., 2003). For this reason, it was decided, in conjunction with the Reuse in Software Engineering (RiSE) members and software and quality engineers of a Brazilian software factory, that the proposed Component Quality Model, similarly to (Goulão et al., 2002b), (Bertoa et al., 2002), would not contemplate this characteristic.

Concurrently, a sub-characteristic had its name changed, as well as its meaning in this new context: the *Installability*, which in the proposed model has the new name of *Deployability*. After developed, the components are deployed (not installed) in an execution environment to make possible their usage by other component-based applications that will be further developed. Through this modification, the understandability of this sub-characteristic becomes clearer to the component context.

Another characteristic that changed its meaning was *Usability*. The reason is that the end-users of components are the application developer and designers that have to build new applications with them, more than the people (end-users) that have to interact with them. Thus, the usability of a component should be interpreted as its ability to be used by the application developer when constructing a software product or a system with it.

Basically, the other characteristics of the model maintain the same meaning for software components than for software products, except for little adaptations that are necessary to bring the ISO/IEC 25010 characteristics definition to the component context.

Besides, another important characteristic was proposed, called *Marketability* (last row of table 5.1). This characteristic expresses the marketing characteristics of a software component and become important to be analyzed in a software component evaluation process, such as:

- **Price:** The cost of the component;
- **Time to market:** The time consumed to make the component available on the market;
- **Targeted market:** The targeted market volume; and

- **Licensing:** The kind of licensing that the software component is available.

This information are important to analyze some factors that bring credibility to the component customers (i.e. developers and designers), for example, the kind of component license is interesting for the costumer analyzed the cost/benefit of buy the component; the target market of a component describes which domains a certain component can be applied; etc.

5.2.2 Quality characteristics that were adapted from ISO/IEC 25010

The previous section presented the major changes, in relation to ISO/IEC 25010, that were introduced in the proposed component quality model. This section presents the sub-characteristics from ISO/IEC 25010 that were maintained in the proposed model, with some adaptation to better reflect the CBSD scenario. The characteristics with their respective sub-characteristics are described next:

Functionality:

- **Suitability:** This sub-characteristic expresses how well the component fits the specified requirements;
- **Accuracy:** This sub-characteristic evaluates the percentage of results obtained with the correct precision level demanded;
- **Interoperability:** The ability of a component to interact with another component (data and interface compatibility);
- **Security:** This sub-characteristic indicates how the component is able to control the access to its provided services; and
- **Compliance:** This sub-characteristic indicates if a component is conforming to any standard (e.g. international standard, certificated in any organization, etc.).

Reliability:

- **Maturity:** This sub-characteristic evaluates the component evolution when it is launched to the market (e.g. number of versions launched to correct bugs, number of bugs corrected, time to make the versions available, etc.);

- **Recoverability:** This sub-characteristic indicates whether the component can handle error situations, and the mechanism implemented in that case (e.g. exceptions);
- **Fault Tolerance:** This sub-characteristic indicates whether the component can maintain a specified level of performance in case of faults; and
- **Compliance:** This sub-characteristic indicates if a component is conforming to any standard (e.g. international standard, certificated in any organization, etc.).

Usability:

- **Understandability:** This sub-characteristic measure the degree of easiness to understand the component (e.g. documentation, descriptions, demos, API's, tutorial, code, etc.);
- **Learnability:** This sub-characteristic measures the time and effort needed to master some specific tasks (e.g. usage, configuration, administration of the component);
- **Operability:** This sub-characteristic measure the ease to operate a component and to integrate the component into the final system; and
- **Compliance:** This sub-characteristic indicates if a component is conforming to any standard (e.g. international standard, certificated in any organization, etc.).

Efficiency:

- **Time Behavior:** This sub-characteristic indicates the ability to perform a specific task at the correct time, under specified conditions;
- **Resource behavior:** This sub-characteristic indicates the amount of the resources used, under specified conditions; and
- **Compliance:** This sub-characteristic indicates if a component is conforming to any standard (e.g. international standard, certificated in any organization, etc.).

Maintainability:

- **Stability:** This sub-characteristic indicates the stability level of the component in preventing unexpected effect caused by modifications;

- **Changeability:** This sub-characteristic indicates whether specified changes can be accomplished and if the component can easily be extended with new functionalities;
- **Testability:** This sub-characteristic measures the effort required to test a component in order to ensure that it complies with its intended function; and
- **Compliance:** This sub-characteristic indicates if a component is conforming to any standard (e.g. international standard, certificated in any organization, etc.).

Portability:

- **Replaceability:** This sub-characteristic indicates whether the component is “backward compatible” with its previous versions; and
- **Adaptability:** This sub-characteristic indicates whether the component can be adapted to different specified environments; and
- **Compliance:** This sub-characteristic indicates if a component is conforming to any standard (e.g. international standard, certificated in any organization, etc.).

5.2.3 Summary

Table 5.2 shows another classification for the proposed component quality model. According to the moment when a sub-characteristic is observed or measured, it can be classified in two kinds: characteristics that are observable at *runtime* (that are discernable at component execution time) and characteristics that are observable during the product *development time* (that are discernable at component development and/or component-based system development).

Table 5.2. The Proposed Component Quality Model, with the sub-characteristics being divided into two kinds: runtime and development time.

Characteristics	Sub-Characteristics (Runtime)	Sub-Characteristics (Development Time)
Functionality	Accuracy Security	Suitability Interoperability Compliance Self-sufficiency
Reliability	Fault Tolerance Recoverability	Maturity
Usability	Configurability	Understandability

		Learnability Operability
Efficiency	Time Behavior Resource Behavior Scalability	
Maintainability	Stability	Changeability Testability
Portability	Deployability	Replaceability Adaptability Reusability

Once the characteristics and sub-characteristics are defined, there must be a way to determine whether a component fulfills them or not. This is achieved through the use of attributes and metrics.

Normally, a quality model consists of four elements (ISO/IEC 25010, 2005): **(i)** characteristics, **(ii)** sub-characteristics, **(iii)** attributes and **(iv)** metrics (Figure 5.2). A quality characteristic is a set of properties of a software product through which its quality can be described and evaluated. A characteristic may be refined into multiple levels of sub-characteristic.



Figure 5.2. Relations among the quality model elements.

An attribute is a measurable physical or abstract property of an entity. A metric defines the measurement method and the measurement scale. The measurement process consists in assigning a number or category to an attribute, according to the type of metric that is associated to that attribute (Square project).

Next, the quality attributes of the CQM will be presented.

5.3 Component Quality Attributes

Last section discussed the general points of the proposed component quality model. This section describes the quality attributes for each sub-characteristic proposed for software components (Alvaro et al., 2005c).

Table 5.3 shows the component quality attributes that are observable at *runtime*. After that, the component quality attributes that are observable at *development time* will be presented.

Table 5.3. Component Quality Attributes for Sub-Characteristics that are observable at *Runtime*.

Sub-Characteristics (Runtime)	Attributes
Accuracy	Correctness
Security	Data Encryption Controllability Auditability
Recoverability	Error Handling
Fault Tolerance	Mechanism availability Mechanism efficiency
Configurability	Effort to configure
Time Behavior	Response time Latency <ul style="list-style-type: none">• Throughput (“out”)• Processing Capacity (“in”)
Resource Behavior	Memory utilization Disk utilization
Scalability	Processing capacity
Stability	Modifiability
Deployability	Complexity level

Next, a brief description of each quality attributes is presented:

Accuracy Sub-Characteristic

Correctness: This attribute evaluates if the component executes as specified by the user requirements

Security Sub-Characteristic

Data Encryption: This attribute expresses the ability of a component to deal with encryption in order to protect the data it handles;

Controllability: This attribute indicates how the component is able to control the access to its provided interfaces;

Auditability: This attribute shows if a component implements any auditing mechanism, with capabilities for recording users access to the system and to its data;

Recoverability Sub-Characteristic

Error Handling: This attribute indicates whether the component can handle error situations, and the mechanism implemented in that case (e.g. exceptions in Java);

Fault Tolerance sub Characteristic

Mechanism availability: This attribute indicates the existence of fault-tolerance mechanisms implemented in the component;

Mechanism efficiency: This attribute measures the real efficiency of the fault-tolerance mechanisms that are available in the component;

Configurability Sub-Characteristic

Effort to configure: This attribute measures the ability of the component to be configured;

Time Behavior Sub-Characteristic

Response time: This attribute measures the time taken since a request is received until a response has been sent;

Latency (the time between the instantiation of a functionality and the time left to obtain the answer)

- **Throughput (“out”):** This attribute measures the output that can be successfully produced over a given period of time;
- **Processing Capacity (“in”):** This attribute measures the amount of input information that can be successfully processed by the component over a given period of time;

Resource Behavior Sub-Characteristic

Memory utilization: The amount of memory needed by a component to operate;

Disk utilization: This attribute specifies the disk space used by a component;

Scalability Sub-Characteristic

Processing capacity: This attribute measures the capacity of the component to support a vast volume of data;

Stability Sub-Characteristic

Modifiability: This attribute indicates the component behavior when modifications are introduced; and

Deployability Sub-Characteristic

Complexity level: This attribute indicates the effort needed to deploy a component in a specified environment.

The quality attributes that are observable during *life cycle* are summarized in Table 5.4. These attributes could be measured during the development of the component or the component-based system, by collecting relevant information for the model.

Table 5.4. Component Quality Attributes for Sub- Characteristics that are observable during *Life cycle*.

Sub-Characteristics (Life cycle)	Attributes
Suitability	Coverage Completeness Pre and Post-conditioned Proofs of Pre and Post-conditions
Interoperability	Data Compatibility Interface Compatibility
Compliance	Standardization Certification
Self-sufficiency	Dependability
Maturity	Volatility Failure removal
Understandability	Documentation availability Documentation readability and quality Code Readability
Learnability	Time and effort to (use, configure, admin and expertise) the component.
Operability	Complexity level Provided Interfaces Required Interfaces Effort to operate
Changeability	Extensibility Customizability Modularity

Testability	Test suite provided Extensive component test cases Component tests in a specific environment Proofs the components tests
Adaptability	Mobility Configuration capacity
Replaceability	Backward Compatibility
Reusability	Domain abstraction level Architecture compatibility Modularity Cohesion Coupling Simplicity

A brief description of each quality attribute is presented next:

Suitability Sub-Characteristic

Coverage: This attribute measures how much of the required functionality is covered by the component implementation;

Completeness: It is possible that some implementations do not completely cover the specified services. This attribute measures the number of implemented operations compared to the total number of specified operations;

Pre-conditioned and Post-conditioned: This attribute indicates if the component has pre- and post-conditions in order to determine more exactly *what* the component requires and *what* the component provides;

Proofs of pre-conditions and post-conditions: This attribute indicates if the pre and post-conditions are formally proved in order to guarantee the correctness of the component functionalities;

Interoperability Sub-Characteristic

Data Compatibility: This attribute indicates whether the format of the data handled by the component is compliant with any international standard or convention (e.g. XML);

Interface Compatibility: This attribute indicates the format/standard that the component provides its interface (e.g. WDSL, etc);

Compliance Sub-Characteristic

Standardization: This attribute indicates if the component conforms to international standards;

Qualifications: This attribute indicates if the component is certified by any internal or external organization;

Self-sufficiency Sub-Characteristic

Dependability: This attribute indicates if the component is not self-sufficiency, i.e. if the component depends on other components to provide its specified services;

Maturity Sub-Characteristic

Volatility: This attribute indicates the average time between different commercial versions/releases;

Failure removal: This attribute indicates the number of bugs fixed in a given component version. The number of bugs fixed in a version (in a period of time) could indicate that the new version is more stable or that the component contains a lot of bugs that will emerge;

Understandability Sub-Characteristic

Documentation availability: This attribute deals with the component documentation, descriptions, demos, APIs and tutorials available, which have a direct impact on the understandability of the component;

Documentation readability and quality: This attribute indicates the quality of the component documentation;

Code Readability: This attribute indicates how easy it is to understand the source code;

Learnability Sub-Characteristic

Time and effort to (use, configure, admin and expertise) the component: This attribute measures the time and effort needed to

master some specific tasks (such as using, configuring, administrating, or expertising the component);

Operability Sub-Characteristic

Complexity level: This attribute indicates the capacity of the user to operate a component;

Provided Interfaces: This attribute counts the number of provided interfaces by the component as an indirect measure of its complexity;

Required Interfaces: This attribute counts the number of interfaces that the component requires from other components to operate;

Effort to operate: This attribute shows the average number of operations per provided interface (operations in all provided interfaces / total of the provided interfaces);

Changeability Sub-Characteristic

Extensibility: This attribute indicates the capacity to extend a certain functionality of the component (i.e. which is the percentage of the functionalities that could be extended);

Customizability: This attribute measures the number of customizable parameters that the component offers (e.g. number of parameters to configure in each provided interface);

Modularity: This attribute indicates the modularity level of the component in order to determine if it is easy or not to modify it, based in its inter-related modules;

Testability Sub-Characteristic

Test suite provided: This attribute indicates whether some test suites are provided for checking the functionality of the component and/or for measuring some of its properties (e.g. performance);

Extensive component test cases: This attribute indicates if the component was extensively tested before being made available to the market;

Component tests in a specific environment: This attribute indicates in which environments or platforms a certain component was tested;

Proofs the components test: This attribute indicates if the component tests were formally proved;

Adaptability Sub-Characteristic

Mobility: This attribute indicates in which containers this component was deployed and to which containers this component was transferred;

Configuration capacity: This attribute indicates the percentage of the changes needed to transfer a component to other environments;

Replaceability Sub-Characteristic

Backward Compatibility: This attribute is used to indicate whether the component is “backward compatible” with its previous versions or not;

Reusability Sub-Characteristic

Domain abstraction level: This attribute measures the component’s abstraction level, related to its business domain;

Architecture compatibility: This attribute indicates the level of dependability of a specified architecture;

Modularity: This attribute indicates the modularity level of the component, if it has modules, packages or if all the source files are grouped in a single bunch;

Cohesion: This attribute measures the cohesion level between the inter-related parts of the component. A component should have high cohesiveness in order to increase its reusability level;

Coupling: This attribute measures the coupling level of the components. A component should have low coupling in order to increase its reusability level; and

Simplicity: This attribute indicates if the component modules are well-defined, concise and well-structured.

Besides the quality characteristics presented, the model is complemented with other kinds of characteristics. These characteristics bring relevant information for new customers and are composed of *Productivity*, *Satisfaction*, *Security* and *Effectiveness*. According to ISO/IEC 25010, these characteristics

are called *Quality in Use* (ISO/IEC 25000, 2005). This is the user's view (i.e. developers or designers) of the component, obtained when they use a certain component in an execution environment and analyze the results according to their expectations. These characteristics show whether the developers or designers can trust a component or not. Thus, *Quality in Use* characteristics are useful to show the component's behavior in different environments.

These characteristics are measured through the customer's feedback. A five-category rating scale is used, ranging from "Very Satisfied" to "Very Dissatisfied". A "Don't Know" option is also included. Using this scale, the *Satisfaction*, *Productivity*, *Security* and *Effectiveness* of the component used by a certain user (developer or designer) can be measured. This user's feedback is very important to the model in order to describe if a certain component is really good in practice, i.e. in a real environment. Of course, this evaluation is subjective, and therefore it must be analyzed very carefully, possibly confronting this information with other facts, such as the nature of the environment and the user's characteristics.

As shown in Figure 5.1., it is needed a kind of measurement for each quality attribute proposed. However, the measurement of those quality attributes described during this chapter will be presented on chapter 6 and Appendix A, which describes the paradigm adopted to define the metrics and gives a set of examples to help the evaluation team during the metrics definition, respectively. The idea is to provide a more flexible way to develop the metrics during the evaluation runtime.

These attributes cover most of the important characteristics that help determining if a component has the desired quality level. However, there are other kinds of information that are important in the process of evaluating a component's quality level, but that were not included in the model because they do not represent quality attributes for software components. Instead, they contain relevant information for a well-defined component evaluation process. These are presented in the next section.

5.4 Other relevant Component Information

In an effective software component evaluation process, some additional information is needed in order to complement the model. Table 5.5 shows the additional characteristics that were identified as being interesting to a software component evaluation process. These characteristics are called *Additional Information* and are composed of: *Technical Information* and *Organization Information*.

Technical Information is important for developers to analyze the actual state of the component (i.e. if the component has evolved, if any patterns were used in the implementation, which kinds of technical support are available for that product, etc.). Besides, it is interesting to the customer that he/she knows who is the responsible for that component, i.e. who maintains that component (e.g. a component created by a software factory CMMI level 5, probably, is more reliable than a component created by an unknown or a new software factory). Thus, the necessity of the *Organization Information* was identified.

Table 5.5. Additional Information.

Additional Information	Technical Information
	<ul style="list-style-type: none">• Component Version• Programming Language• Patterns Usage• Lines of Code• Technical Support
	Organization Information
	<ul style="list-style-type: none">• CMMi / MPS.br Level• Organization's Reputation

The *Additional Information* provides relevant component information to the model. The main concern is that these characteristics are the basic information to whatever kind of components available in the market.

5.5 Summary

This chapter presented an overview of the proposed software component quality framework, showing their importance to the whole framework as well. Besides, the proposed Component Quality Model was also presented, showed its characteristics, sub-characteristics and the quality attributes. The relevant characteristics, which were not included into the model because they do not

represent quality attributes for software components was also presented. However, they complement CQM with relevant information, helping in the software component evaluation process. Figure 5.3 shows the summary of the proposed CQM.

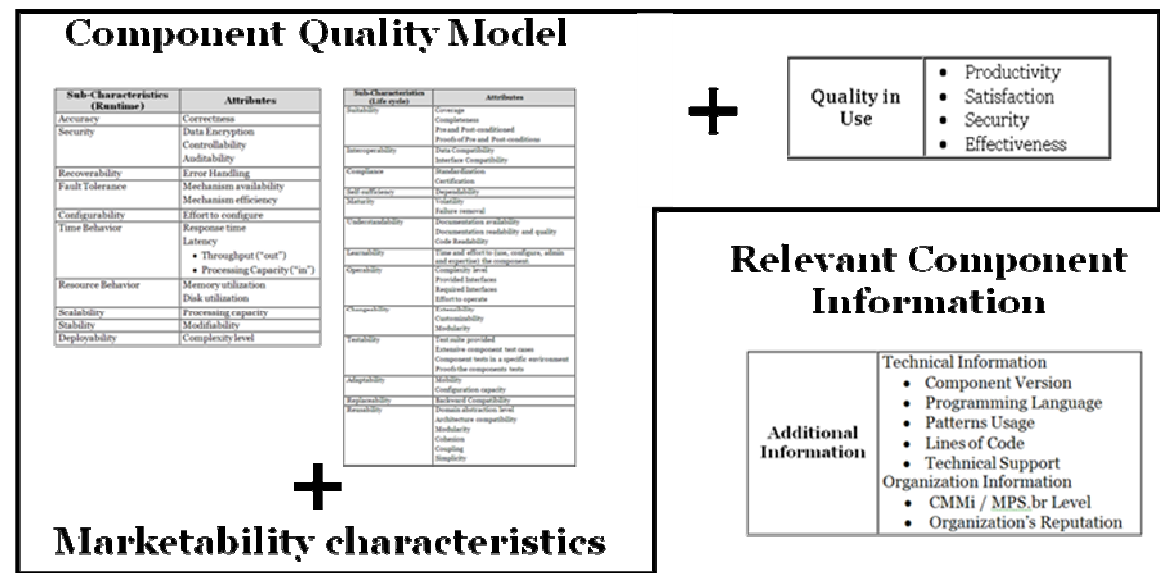


Figure 5.3. Summary of the CQM.

A formal case study using the CQM was developed in order to provide the first insights of its usability and viability to evaluate software components quality. More information about it can be found at (Alvaro et al., 2006c), (Alvaro et al., 2006d).

Finally, more detailed information about the other modules will be presented in the next chapters.

6

Evaluation Techniques Framework and Metrics Framework

After the Component Quality Model (CQM) has been defined, it was necessary to establish evaluation techniques to evaluate each component quality characteristic. Perhaps not all the selected quality characteristics and sub-characteristics proposed need to be evaluated with the same degree of thoroughness and depth for all types of software components. Nobody would expect the same effort to be allocated to the component evaluation of a railway signal system, and a component from a computer game system. To ensure this flexibility, the evaluation should be level-oriented. In this way, different evaluation levels must be used in order to provide specialized services for each kind of software components distributed on different domains and risk-levels, providing confidence in the quality of a software component in these domains.

On the other hand, as with any engineering discipline, software development requires a measurement mechanism for feedback and evaluation. Measurement is a mechanism for creating a corporate memory and an aid in answering a variety of questions associated with the enactment of any software process (Basili et al., 1994). Measurement is important in any software engineering area in order to provide data to track the efficiency and efficacy of the process analyzed.

In this way, a Software Component Evaluation Techniques Model (SCETM) was defined (Alvaro et al., 2007b) and will be presented next. After that, the Metrics Framework will be presented in order to provide insights to define the metrics to measure the evaluation techniques presented in this chapter and also metrics to measure the whole framework.

6.1 A Software Component Evaluation Techniques Model (SCETM)

The model is constituted of evaluation levels where the quality of the components can be assure. There are five levels (they constitute a hierarchy), which identify the depth of the evaluation. Evaluation at different levels gives different degrees of confidence in the quality of the software component and the component could increase its level of reliability and quality as it evolves. Thus, each company/customer decides which level is better for evaluating its components, analyzing the cost/benefits of each level. The closer to the last level, the higher is the probability that the component is trustable, contains a consistent documentation and can be easily reused.

Thus, there are five levels which form an increasing set of evaluation requirements, where SCETM 1 is the lowest level and SCETM 5 is the highest level. The evaluation level defines the depth or thoroughness of the evaluation. Therefore evaluation at different levels gives different degrees of confidence in the quality of the software component.

For instance, the level SCETM 5 contains more rigorous evaluation techniques (requiring a high amount of time and effort to execute the evaluation) which are applied to give more confidence to the software component. On the other hand, as you decrease the SCETM levels the techniques used are less rigorous and, consequently, less effort is applied during the evaluation.

There are two different ways to decide about the evaluation: **(i)** the component can be evaluated executing all techniques from one specific level (e.g. component evaluation using level SCETM 2) and; **(ii)** the evaluation levels can be chosen independently for each characteristic (i.e. for functionality the component can be evaluated using the techniques from level SCETM 1; for reliability those techniques from level SCETM 3; for usability those techniques from level SCETM 4 and so on). The idea is to provide more flexibility during the selection levels too, in order to facilitate the model usage and accessibility.

Table 6.1 gives some indication as to which level a given software component should be evaluated. Each column of Table 6.1 represents different layers that the software component should be considered when evaluating its

potential damage and related risks. The level of damage in each layer is the first guideline used to decide which SCETM level is more interesting for the organization; the important aspects are those related to environment, to safety/security and to economy. However, these are mere guidelines, and should not be considered as a rigid classification scheme. Those few guidelines were based on (Boegh et al., 1993), (Solingen, 2000), (ISO/IEC 25000, 2005) and extended to the component context.

Table 6.1. Guidelines for selecting evaluation level.

Level	Environment	Safety/Security	Economic
SCETM 1	No damage	Few material damage; No specific risk	Negligible economic loss
SCETM 2	Small/Medium damage properly	Few people disabled	Few economic loss
SCETM 3	Damage properly	Large number of people disabled	Significant economic loss
SCETM 4	Recoverable environment damage	Threat to human lives	Large economic gross
SCETM 5	Unrecoverable environmental damage	Many people killed	Financial disaster

A set of appropriate evaluation techniques were defined. Relevant works from literature that propose evaluation techniques for software product were analyzed (Boegh et al., 1993), (Solingen, 2000), (Tian, 2004), (ISO/IEC 25000, 2005), (TMMi, 2008) and the experience of one software quality specialist of Federal University of Pernambuco and a set of software quality/system engineers from a Brazilian software factory helped during this definition. Afterwards, the feedback of relevant researchers on CBD from Mälardalen University¹² (Department of Computer and Electrical Engineering) from Sweden and a specialist from ABB Company¹³ was very important to the model evolution.

¹² <http://www.mdh.se/ide>

¹³ <http://www.abb.com>

Moreover, a set of works from the literature about each single technique was analyzed in order to identify the real necessity of those evaluation techniques. In this way, the following techniques were proposed in the model:

- **SCETM I**

- **Documentation Analysis:** it focus on analyze the documents available in order to properly use a component in the way it was intended (Kallio et al., 2001), (Kotula, 1998), (Lethbridge et al., 2003), (Taulavuori et al., 2004);
- **Suitability analysis:** it focus on measures how well the component reliability fits the specified requirements (Wohlin & Regnell, 1998), (Hamlet et al., 2001), (McGregor et al., 2003);
- **Effort to configure analysis:** if focus on measures the ability of the component to be configured (Brownsword et al., 2000), (Bertoa et al., 2002);
- **Accuracy analysis:** it focus on evaluates the percentage of results obtained with the correct precision level demanded (Bertoa et al., 2002);
- **Effort for operating:** it focus on measures the complexity to use the component, through its interfaces available, and achieve the expected flow of execution (Brownsword et al., 2000), (Bertoa et al., 2002), (Bertoa & Vallecillo, 2004);
- **Customizability analysis:** if focus on measures the kind of parameterization available for each interfaces of the component (Brownsword et al., 2000);
- **Extensibility analysis:** if focus on measure the complexity (i.e. the effort spent) to extend the component's functionalities (Brownsword et al., 2000), (Bertoa et al., 2002), (Bertoa & Vallecillo, 2004);
- **Component execution in specific environments analysis:** it focus on analyze the complexity to deploy a component in a specific environment (Brownsword et al., 2000);
- **Cohesion, Coupling, Modularity and Simplicity analyses:** if focus on analyze the reusability level of the component (Caldiera & Basili, 1991), (Gui & Scott, 2007);

- **Cohesion of the documentation with the source code analysis:** it focus on analyze the documentation compliance with the source code, relating to portability characteristic (Kallio et al., 2001), (Bay and Pauls, 2004);
- **Cohesion of the documentation with the source code analysis:** it intends to analyze the compliance of the documentation provided and the source code of the component in order to use it properly (Kallio et al., 2001), (Kotula, 1998), (Lethbridge et al., 2003), (Taulavuori et al., 2004).
- **SCETM II**
 - **Functional Testing (black box):** it focus on the validation of required functional features and behaviors of software component from an external view (Beydeda and Gruhn, 2003), (Gao et al., 2003);
 - **Unit Test:** The primary goal of unit testing is to take the smallest piece (functionality) of a software component, isolate it from the remainder of the code, and determine whether it behaves exactly as you expect. Each unit is tested separately before integrating them into the whole component. Unit testing has proven its value in that a large percentage of defects are identified during its use (Beydeda and Gruhn, 2003).
 - **Regression Test:** it aims to ensure that a modified component still meets the specifications validated before modification (Beydeda and Gruhn, 2003), (Gao et al., 2003);
 - **Programming Language Facilities (Best Practices):** if focus on analyze if the component was developed using the best practices/standards/patterns found on literature that is related to the programming language in question;
 - **Maturity analysis:** if focus on evaluates the component evolution when it is launched to the market (e.g. number of versions launched to correct bugs, number of bugs corrected, time to make the versions available, etc.) (Brownsword et al., 2000);
 - **Inspection of the provided and required interfaces:** it focuses on a systematic approach to examining the interfaces of the

component. Such an examination's goal is to assess the quality of the required and provided interface of a component (Beugnard et al., 1999), (Reussner, 2003), (Parnas and Lawford, 2003);

- **Evaluation measurement (latency analysis):** it focus on the time between the instantiation of a functionality and the time left to obtain the answer in order to measure each component functionalities efficiency (Bertoa et al., 2002), (Brownsword et al., 2000);
- **Inspection of Documents:** it focuses on a systematic approach to examining a document in detail. Such an examination's goal is to assess the quality of the software component documents in question (Fagan, 1976), (Parnas and Lawford, 2003);
- **Deployment analysis:** it validates a component to see if it can be successfully deployed in its new context and operation environment for a specific project (Gao et al., 2003);
- **Backward compatibility:** it focus to analyze if the component is “backward compatible” with its previous versions in order to guarantee its compatibilities between different versions (Bertoa et al., 2002), (Brownsword et al., 2000).
- **SCETM III**
 - **Component Test:** it exercises all functionalities of a component in order to evaluate its compliance with their specification (Freedman, 1991), (Councill, 1999), (Gao et al., 2003), (Beydeda and Gruhn, 2003);
 - **Inspection of Documents:** as described in last level, it focuses on a systematic approach to examining a document in detail. Such an examination's goal is to assess the quality of the software component documents in question (Fagan, 1976), (Parnas and Lawford, 2003);
 - **Fault tolerance analysis:** it focus on analyze the existent (if any) fault tolerance mechanism existent on the component in order to guarantee the error treatment in cause of fault (Bertoa et al., 2002), (Brownsword et al., 2000);

- **Code and component's interface inspection:** it focuses on a systematic approach to examining an interface in detail. Such an examination's goal is to assess the quality of the interfaces provided and required by a software component (Beugnard et al., 1999), (Reussner, 2003), (Parnas and Lawford, 2003);
- **Performance Tests:** it evaluates and measures the performance of a component in a new context and operation environment to make sure that it satisfies the performance requirements (Gao et al., 2003);
- **Code metrics and programming rules:** it focus on collect a set of metrics from a component in order to analyze it and verify the programming language rules presents in source code (Brownsword et al., 2000), (Cho et al., 2001);
- **Static Analysis:** it focuses on checks the component errors without compiling/executing it through a set of tools that support it (Brownsword et al., 2000);
- **Conformity to programming rules:** it focuses on analyze if the rules related to a certain programming languages was adopted and used during the component development.
- **SCETM IV**
 - **Structural Tests (white-box):** it focus on validation of program structures, behaviors, and logic of component from an internal view (Beydeda and Gruhn, 2003), (Gao et al., 2003);
 - **Code inspection:** it focuses on a systematic approach to examining a source code in detail. Such an examination's goal is to assess the quality of the software component codification in question (Fagan, 1976), (Parnas and Lawford, 2003);
 - **Reliability growth model:** it focuses on discover reliability deficiencies through testing, analyzing such deficiencies, and implementation of corrective measures to lower the rate of occurrence. Some of the important advantages of the reliability growth model include assessments of achievement and projecting the product reliability trends (Wohlin & Regnell, 1998), (Hamlet et al., 2001), (McGregor et al., 2003);

- **Analysis of the pre and post-conditions of the component:** it focus on analyze/inspect the pre and post-condition of the component in order to verify if the provided/required services is compliance with the specified conditions defined on the component (Beugnard et al., 1999), (Reussner, 2003);
- **Time to use analysis:** it focus on measures the time and effort needed to master some specific tasks (such as using, configuring, administrating, or expertising the component) (Brownsword et al., 2000);
- **Algorithmic complexity:** Algorithmic complexity quantifies how complex a component is in terms of the length of the shortest computer program, or set of algorithms, need to completely describe the component solution. In other words, it is how small a model of a given component is necessary and sufficient to capture the essential patterns of that component. Algorithmic complexity has to do with the mixture of repetition and innovation in a complex component. At one extreme, a highly regular component can be described by a very short algorithm (Cho et al., 2001);
- **Analysis of the component development process:** it focus on inspect the whole CBD process in order to find any gap, inconsistence, fault, etc during the component life-cycle development (Brownsword et al., 2000);
- **Environment constraints evaluation:** it focus on analyze the whole environment that the component will be deployed in order to collect constraints that could affect any quality attribute during its evaluation (Choi et al., 2008) ;
- **Domain abstraction analysis:** it focus on analyze how abstract is the solution implemented in a certain component from its principals related domains. Thus, the component may be a higher reuse level (Bay and Pauls, 2004), (Gui & Scott, 2007).
- **SCETM V**
 - **Formal Proof:** Formal methods are intended to systematize and introduce rigor into all the phases of software component development. The idea is try to avoid overlooking critical issues,

provides a standard means to record various assumptions and decisions, and forms a basis for consistency among many related activities. By providing precise and unambiguous description mechanisms, formal methods facilitate the understanding required to coalesce the various phases of software development into a successful endeavor (Boer et al., 2002);

- **User mental model:** it focuses on the mental understanding of what the component is doing for the user. Mental models are the conceptions of a component that develop in the mind of the user. Mental models possess representations of objects or events in a component and the structural relationships between those objects and events. Mental models evolve inductively as the user interacts with the component, often resulting in analogical, incomplete, or even fragmentary representations of how the component works (Farooq & Dominick, 1988);
- **Performance profiling analysis:** it focuses on investigation of a component's behavior using information gathered (i.e. it is a form of dynamic program analysis, as opposed to static code analysis). The usual goal of performance analysis is to determine which sections of a component should be optimize – usually either to increase its speed or decrease its memory requirement. (Bertolino & Mirandola, 2003), (Chen et al., 2005);
- **Traceability evaluation:** it refers to the extent of its built-in capability that tracks functional operation, component attributes, and behaviors (Gao et al., 2003);
- **Component Test Formal Proof:** Formal methods are intended to systematize and introduce rigor into all the phases of software component development, in this case on component test phase. The idea is try to avoid misunderstanding during the component test phase and guarantee that each single part of the component was tested (Boer et al., 2002);
- **Analysis of the component's architecture:** if focus on examine the architecture defined to develop the component solution, looking for analyzes if the actual architecture impact any

quality attribute level of the component (Kazman et al., 2000), (Choi et al., 2008).

Those selected techniques bring, each one for each specific aspects, a kind of quality assurance to software components. The establishment of what each level is responsible for is very valuable to the evaluation team during the definition of the evaluation techniques for those levels. In other words, the evaluation team knows what is more important to consider during the component evaluation and try to correlate these necessities with the evaluation level more appropriated. The intention was to build a model where the techniques selected to represent each level should complement each other in order to provide the quality degree needed for each SCETM level. The SCETM levels and the evaluation techniques are presented in Table 6.2. Additionally, in order to understand the meaning of each level, next some description is presented:

- **SCETM 1:** the first level intends to investigate if the component does what is described in its documentation, its reusability level, the effort to use and maintain the component and its correct execution in defined environments. The aim of this level is the compatibility between the documentation and the component's functionalities. For that, some kind of analysis in documentation, environment and in the component should be done in order to guarantee that it is correctly defined, implemented and described;
- **SCETM 2:** the second level worries about the correct execution of the component, applying a set of test techniques, inspecting the documentation better, if the component uses best practices in the chosen programming language and to evaluate the correct component deployment. The techniques of this level analyze the correct execution of the component. For that, a set of inspections should be done in order to evaluate if the component can be deployed correctly in the environment specified in its documentation and, successively, it can be correct used/instantiated;
- **SCETM 3:** the main interests of this level are to evaluate how the component can avoid faults and errors, analyzing the provided and

required interfaces looking for correct design and to evaluate a set of programming rules. The aim of this level is to analyze if the component can avoid or tolerate faults and errors during its execution. For that a set of analysis should be done and several rules must be checked in order to guarantee that the component, if happened some fault/errors, can administrate theses faults/errors through some kind of implementation or some kind of available techniques, among others;

- **SCETM 4:** in this level the source-code of the component is needed in order to inspect it more precisely. The code is inspected and tested, the provided and required interfaces are revisited and the algorithm complexity is examined in order to prove its performance too. An interesting aspect of this level is the analysis of the Component-Based Development (CBD) process, looking for possibilities of improving the CBD process adopted. The aim of this level is to assure the component's performance. For that some low techniques should be applied in order to try finding any unnecessary complexity in the component implementation looking for achieving the maximum quality, performance and reliability; and
- **SCETM 5:** the last level is considered the formal proof of the component functionalities and reliability. The architecture and the traceability are also examined in this level. Here, the idea is to achieve the highest level of trust that is possible. As a result, the techniques in this level tend to be the most costly and time consuming. Hence, the ROI (Return On Investment) analysis on this level is very important. The aim of this level is to increase the trust in the component as much as possible. For that, it is necessary to guarantee that the formalism presented on the component is corrected and should be proved through a set of verifications accomplished in the specification.

Table 6.2. Software Component Evaluation Techniques Model.

Characteristics	SCETM I	SCETM II	SCETM III	SCETM IV	SCETM V
Functionality	<ul style="list-style-type: none"> • Documentation Analysis 	<ul style="list-style-type: none"> • Functional Testing (black box), Unit Test, Regression Test (if possible) 	<ul style="list-style-type: none"> • Component Test • Inspection of Documents 	<ul style="list-style-type: none"> • Structural Tests (white-box) with coverage criteria and code inspection 	<ul style="list-style-type: none"> • Formal Proof
Reliability	<ul style="list-style-type: none"> • Suitability analysis 	<ul style="list-style-type: none"> • Programming Language Facilities (Best Practices) • Maturity analysis 	<ul style="list-style-type: none"> • Fault tolerance analysis 	<ul style="list-style-type: none"> • Reliability growth model 	<ul style="list-style-type: none"> • Formal Proof
Usability	<ul style="list-style-type: none"> • Documentation analysis (Use Guide, architectural analysis, etc) • Effort to Configure analysis 	<ul style="list-style-type: none"> • Inspection of the provided and required interfaces 	<ul style="list-style-type: none"> • Code and component's interface inspection (correctness and completeness) 	<ul style="list-style-type: none"> • Analysis of the pre and post-conditions of the component • Time to use analysis 	<ul style="list-style-type: none"> • User mental model
Efficiency	<ul style="list-style-type: none"> • Accuracy analysis 	<ul style="list-style-type: none"> • Evaluation measurement (latency analysis) 	<ul style="list-style-type: none"> • Performance Tests 	<ul style="list-style-type: none"> • Algorithmic complexity 	<ul style="list-style-type: none"> • Performance profiling analysis
Maintainability	<ul style="list-style-type: none"> • Effort for operating • Customizability analysis • Extensibility analysis 	<ul style="list-style-type: none"> • Inspection of Documents 	<ul style="list-style-type: none"> • Code metrics and programming rules • Static Analysis 	<ul style="list-style-type: none"> • Analysis of the component development process 	<ul style="list-style-type: none"> • Traceability evaluation • Component Test Formal Proof
Portability	<ul style="list-style-type: none"> • Component execution in specific environments analysis • Cohesion, Coupling, Modularity and Simplicity analyses • Cohesion of the documentation with the source code analysis 	<ul style="list-style-type: none"> • Deployment analysis • Backward compatibility 	<ul style="list-style-type: none"> • Conformity to programming rules 	<ul style="list-style-type: none"> • Environment constraints evaluation • Domain abstraction analysis 	<ul style="list-style-type: none"> • Analysis of the component's architecture

One of the main concerns during SCETM definition is that the levels and the evaluation techniques selection must be appropriated to completely evaluate the quality attributes proposed on the CQM, presented in chapter 5. This is achieved through a mapping of the *Quality Attributes X Evaluation Technique*. For each quality attribute proposed in the CQM, it is interesting that at least one technique is proposed in order to cover it completely, also being capable of measuring it properly. Tables 6.3a, 6.3b and 6.3c show this matching between the CQM quality attributes and the proposed SCETM evaluation techniques.

Theses Tables show that the main concern is not to propose a big amount of isolated techniques, but to propose a set of techniques that are essential for measuring each quality attribute, complementing each other and, thus, becoming useful to compose the Evaluation Techniques Framework.

Table 6.3a Component Quality Attributes X Evaluation Techniques

Sub-Characteristics	Quality Attributes	Evaluation Techniques
Suitability	Coverage	• Documentation Analysis
	Completeness	• Documentation Analysis
	Pre and Post-conditions	• Code Inspection
	Proofs of Pre and Post-conditions	• Formal Proof
Accuracy	Correctness	<ul style="list-style-type: none"> • Functional Testing (black box), Unit Test, Regression Test (if possible) • Functional Tests (white-box) with coverage criteria
Interoperability	Data Compatibility	<ul style="list-style-type: none"> • Inspection of Documents • Code Inspection
Security	Data Encryption	<ul style="list-style-type: none"> • System Test • Code Inspection
	Controllability	<ul style="list-style-type: none"> • System Test • Code Inspection
	Auditability	<ul style="list-style-type: none"> • System Test • Code Inspection
Compliance	Standardization	• Inspection of Documents
	Certification	• Inspection of Documents
Self-contained	Dependability	<ul style="list-style-type: none"> • Documents Inspection • Code Inspection

Table 6.3b Component Quality Attributes X Evaluation Techniques

Sub-Characteristics	Quality Attributes	Evaluation Techniques
Maturity	Volatility	• Suitability analysis
	Failure removal	• Maturity analysis
Recoverability	Error Handling	<ul style="list-style-type: none"> • Programming Language Facilities (Best Practices) • Error Manipulation analysis • Reliability growth model • Formal Proof
Fault Tolerance	Mechanism available	• Suitability analysis
	Mechanism efficiency	<ul style="list-style-type: none"> • Programming Language Facilities (Best Practices) • Fault tolerance analysis • Reliability growth model • Formal Proof
Understandability	Documentation available	• Documentation analysis (Use Guide, architectural, etc)
	Documentation readability and quality	• Documentation analysis (Use Guide, architectural, etc)
	Code Readability	• Code and component's interface inspection (correctness and completeness)
Configurability	Effort for configure	• Effort to Configure analysis
Learnability	Time and effort to (use, configure, admin and expertise) the component.	• Time to use analysis
Operability	Complexity level	• User mental model
	Provided Interfaces	• Inspection of the interfaces
	Required Interfaces	• Inspection of the interfaces
	Effort for operating	• User mental model
Time Behavior	Response time	• Accuracy analysis
	Latency a. Throughput ("out")	• Evaluation measurement (latency analysis)
	b. Processing Capacity ("in")	• Evaluation measurement (latency analysis)
Resource Behavior	Memory utilization	• Tests of performance
	Disk utilization	• Tests of performance

Table 6.3c Component Quality Attributes X Evaluation Techniques

Sub-Characteristics	Quality Attributes	Evaluation Techniques
Scalability	Processing capacity	<ul style="list-style-type: none"> • Tests of performance • Algorithmic complexity • Performance profiling analysis
Stability	Modifiability	<ul style="list-style-type: none"> • Code metrics and programming rules • Inspection of Documents • Static Analysis
Changeability	Extensibility	<ul style="list-style-type: none"> • Effort for operating • Extensibility analysis
	Customizability	<ul style="list-style-type: none"> • Effort for operating • Customizability analysis
	Modularity	<ul style="list-style-type: none"> • Code metrics and programming rules
Testability	Test suit provided	<ul style="list-style-type: none"> • Analysis of the test-suite provided (if exists)
	Extensive component test cases	<ul style="list-style-type: none"> • Analysis of the component development process
	Component tests in a specific environment	<ul style="list-style-type: none"> • Traceability evaluation
	Proofs the components test	<ul style="list-style-type: none"> • Component Test Formal Proof
Deployability	Complexity level	<ul style="list-style-type: none"> • Component execution in specific environments analysis • Deployment analyses • Environment constraints evaluation
Replaceability	Backward Compatibility	<ul style="list-style-type: none"> • Backward compatibility analysis
Reusability	Domain abstraction level	<ul style="list-style-type: none"> • Cohesion of the documentation with the source code analysis • Domain abstraction analysis
	Architecture compatibility	<ul style="list-style-type: none"> • Conformity to programming rules • Analysis of the component's architecture
	Modularity	<ul style="list-style-type: none"> • Modularity analyses
	Cohesion	<ul style="list-style-type: none"> • Cohesion analyses
	Coupling	<ul style="list-style-type: none"> • Coupling analyses
	Simplicity	<ul style="list-style-type: none"> • Simplicity analyses

Those evaluation techniques are the starting point where the software component evaluator sets up her/his work. The idea is to incrementally increase the appropriate techniques used during the previous component evaluation and through the evaluation feedback too. Thus, the SCETM will be composed of a set of techniques available to use and the software evaluator will decide which technique is better for each component evaluation, depending on the programming language, component domain, deployment environment, domain-risk level, among other factors. It is very interesting that the evaluation team gives its feedback about the SCETM techniques in order to increase the amount and quality of those techniques proposed in each level. The same idea is applicable to the *Guidelines for selecting evaluation level* (Table 6.1), where the guidelines should be improved through evaluations feedback.

Also, some initial guidance for estimating the cost of an evaluation can be given. The actual cost of evaluating will depend on the level of the evaluation, the size of the component, the amount and quality of the available documentation, special requirements demanded by the customer, laws and regulations, and possibly other factors. No empirical data to prove these factors is available yet, however these factors should be considered as a starting point before initiating a software component evaluation.

Based on the guidelines for selecting the evaluation level (Table 6.1) and the costs/benefits (some brief guidance cited above), the costumer can choose the level that the component will be evaluated.

Each technique can be executed using a different kind of process, methods and tools, which depends, basically, on the programming language and deployment environment. The evaluator is responsible for that decision and is very valuable if he/she stores the processes, methods and tools used to execute the techniques selected (this could be stored in a simple table describing each *evaluation techniques selected X process/methods/tools defined X its usefulness*). Thus, in the future, the evaluation team will have a new table describing which processes, methods or tools were used for each evaluation technique during previous evaluation and, probably, help him/her in that new selection.

6.2 Metrics Framework

According to Basili et al. (Basili et al., 1994), the measurement must be defined in a top-down fashion. It must be focused, based on goals and models. A bottom-up approach will not work because there are many observable characteristics in software (e.g., time, number of defects, complexity, lines of code, severity of failures, effort, productivity, defect density), but which metrics one uses and how one interprets them is not clear without the appropriate models and goals to define the context.

There are a variety of mechanisms for defining measurable goals that have appeared in the literature: the Software Quality Metrics approach (Boehm et al., 1976), (McCall et al., 1977), the Quality Function Deployment approach (Kogure & Akao, 1983), the Goal Question Metric approach (Basili et al., 1994), (Basili, 1992), (Basili & Rombach, 1988), (Basili & Selby, 1984), (Basili & Weiss, 1984), the Practical Software Measurement (PSM) (McGarry et al., 2002) and the ISO/IEC 15939 (ISO/IEC 15939, 2007). However, in this work the Goal-Question-Metric (GQM) approach was adopted, which was the same technique proposed to be used in ISO/IEC 25000 looking for track the software product properties.

The Goal Question Metric (GQM) approach is based upon the assumption that for an organization to measure in a purposeful way it must first specify the goals for itself and its projects, then it must trace those goals to the data that are intended to define those goals operationally, and finally provide a framework for interpreting the data with respect to the stated goals. Thus it is important to make clear, at least in general terms, what informational needs the organization has, so that these needs for information can be quantified whenever possible, and the quantified information can be analyzed in order to achieve the target goals of a measurement.

The GQM is a measurement model divided into three levels:

1. **GOAL:** A goal is defined for an object, for a variety of reasons, with respect to various models of quality, from various points of view, relative to a particular environment;

2. **QUESTION:** A set of questions is used to characterize the way the assessment/achievement of a specific goal is going to be performed based on some characterizing model. Questions try to characterize the object of measurement (product, process, resource) with respect to a selected quality issue and to determine its quality from the selected viewpoint; and
3. **METRIC:** A set of data is associated with every question in order to answer it in a quantitative way.

The intention is the evaluation consider, during the metrics definition, a set of factors in which could improve the collected results, such as: Meaning of the Metric, Costs and Complexity to measure, Repeatability, Reproductively, Validity, Objectivity and Impartially. Those factors are essential during the elaboration of the metrics using the GQM technique.

In this way, the GQM will support the evaluation team during the definition of the metrics in order to track the properties of the quality attributes described on CQM, the evaluation techniques presented on SCETM as well as the whole component evaluation process (that will be presented on chapter 7). An important aspect should be considered: the complexity to obtain the data of each metric proposed and if the selected metric can represent completely the information required by the evaluator.

Based on the modules of the Software Component Quality Framework, some examples of usage will be described in order to provide insights to the team evaluation during the *Define GQM step* (vide chapter 7 and the first activity of the component evaluation process). The examples will be divided in three steps: **(i)** the metrics to track the properties of the CQM; **(ii)** the metrics to track the properties of the evaluation techniques presented on SCETM; and **(ii)** the metrics to track the properties of the component evaluation process. However, only few examples of how using the GQM to measure those properties will be presented in this chapter. The Appendix A will present more usage examples (at least, one for each quality attribute, some of them for evaluation techniques and some of them to track the evaluation process).

Thus, the main goal of this section is to provide some metrics definition to guide the team evaluation once there is a complete absence of metrics that could

help evaluate software component quality attributes objectively available on literature (Bertoa et al., 2006). However, an important aspect when defining metrics is to be more objective as possible, even so many times it is very difficult to do so because so many quality attributes are, by definition, subjective.

Objective measures are those that depend only on the object of study and possibly some physical apparatus. In contrast, subjective measures are the product of some mental activity. While this distinction is clear, an illustration can help explain some less than obvious subsidiary points.

Consider, for example, measuring the understandability of a computer program. A candidate objective measure of this quality is “*cyclomatic complexity*,” which is defined as a numerical measure derived from the structure of a computer program (in particular, control flow). A candidate subjective measure might be a statement posed to a human subject, such as “this code is understandable,” with a numerical measure derived from responses to this statement ranging from “strongly agree” to “strongly disagree.”

The virtue of objectivity is that the measure is repeatable. But repeatability does not suggest anything about other important qualities of the measure such as reliability. It is not clear whether *cyclomatic complexity* is more reliable than the proposed subjective measure, and, in fact, there is reason to think just the opposite (Wallnau, 2004).

In this way, the following metrics are defined, as much as possible, in an objective way. However, there are a set of metrics that depends of the evaluation context, evaluation team and the data that could be collected during the evaluation, becoming subjective metrics (one example is the *Effort to Configure* quality attribute; if you don’t have empirical data to classify how much effort is considered low, medium or high to configure certain components, it is needed the feeling/experience/knowledge of the evaluation team within the component’s and environment’s context).

6.2.1 Metrics to track the CQM Properties

Chapter 5 presented the Component Quality Model (CQM) with its related characteristics, sub-characteristics and quality attributes. Now, the metrics to measure those quality items defined on chapter 5 is presented.

A few examples of metrics usage will be presented (in order to show how to define the metrics using GQM) and on Appendix A, at least, one example of metrics for each quality attribute will be described. However, it is important to highlight that those metrics must be defined in a component evaluation context. These one presented here and on Appendix A are only to guide the team evaluation during the definitions of the metrics in the first's component evaluation process.

For example, for **Accuracy Sub-Characteristic** the following metric could be applied:

Functionality	
Sub-Characteristic	Accuracy
Quality Attribute	Correctness
Goal	Evaluate the percentage of the results that were obtained with precision
Question	Based on the amount of tests executed, how many test results return with precision?
Metric	Precision on results / Amount of tests
Interpretation	$0 \leq x \leq 1$; which closer to 1 is better

An interesting aspect here is that the evaluation team defines the *Interpretation* of the metrics definition. Thus, the collection and analysis of that metrics become more feasible, repeatable, reproducible and easier to understand in a way that the whole team knows how the attribute was collected.

A subjective metrics, for example, the **Understandability Sub-Characteristic** could be measure using the following metric:

Usability	
Sub-Characteristic	Understandability
Quality Attribute	Document available
Goal	Analyze the documentation availability.
Question	How many documents are available to understand the component functionalities?
Metric	Number of documents
Interpretation	As higher and with quality it is better but it depends of the component complexity, domain, etc.

As shown, it is very complex to measure how understandable is a document. In this way, the metric provided is subjective and it is very dependable of the evaluation team (skills, knowledge, etc.).

6.2.2 Metrics to track the Evaluation Techniques Properties

With the same objective of the last section, now a few metrics to track the properties of the evaluation techniques of the SCETM, described in this chapter, will be provided. Different from last section, each technique proposed here can be measured in different ways and complexity, using different tools, techniques, methods and processes. Thus, the evaluation team should define with a degree of thoroughness which option is more interesting to measure each evaluation technique proposed. Here, some recognized tools or methods from literature will be used as basis, considering that the software components to be evaluated were developed using Java as programming language.

Functionality	
Quality Attribute	Response Time
SCETM level	I
Technique	Accuracy Analyzes using TPTP tool ¹⁴
Goal	Evaluate the percentage of the time taken between a set of invocations
Question	Is the tool efficient to measure the response time of this kind of component?
Metric	Analysis of the results and coverage of the tool
Interpretation	<p>Defines the applicability of the tool to measure this quality attributes. If this tool can measure efficient the response time of a set of invocations, it is good. On the other hand, if it is not enough to evaluate some functions other tool should be use to complement or to substitute this one.</p> <p>If this tool is good to evaluate the component, it intend to analyze how much results are generated with the expected accuracy and the formula could be:</p> $\frac{\text{Number of results with accuracy}}{\text{Number of results generated}}$ <p>0 <= x <= 1; which closer to 1 is better</p>

As shown previously, one possible way to measure the quality attribute *Response Time* using the *Accuracy Analysis* technique could be through the Test & Performance Tools Platform Project (TPTP) tool. However, the team

¹⁴ Eclipse Test & Performance Tools Platform Project (TPTP) – <http://www.eclipse.org/tptp>

evaluation must know the tool usage in order to achieve its best utilization and efficiency in measuring the component quality.

6.2.3 Metrics to track the Evaluation Process Properties

A consistent and good evaluation results can only be achieved by following a high quality and consistent evaluation process (Beus-Dukic et al., 2003). However, to assure the efficiency and efficacy of the process, it is important to define some metrics. The idea is to obtain feedback from those metrics in order to improve the activities and steps proposed to evaluate the component quality (that will be presented on chapter 7). Next, two metrics that could be used for this purpose is presented.

Component Evaluation Process	
Goal	Adequately evaluate the software component
Question	Can the evaluation team evaluated everything they planned to execute using the documents developed during the process activities?
Metric	Total documented functionalities / Total component functionalities (or Total measurement accomplished)
Interpretation	$0 \leq x \leq 1$; which closer to 1 is better.

Component Evaluation Process	
Goal	To analyze the usability of the templates provided
Question	Have the template helped during the evaluation development?
Metric	Evaluation team feedback
Interpretation	If the template helped during the evaluation development, it is good; if not, it should be adapted to improve the time of the component evaluation process.

The metrics presented above are specific to measure certain properties of the evaluation process. However, the team evaluation should define metrics as much as they think interesting, in order to measure the process capability to evaluate the software component quality.

6.3 Summary

This chapter presented the Evaluation Techniques Framework and the Metrics Framework proposal. The idea is that the components used in applications with different risks-level must also be evaluated differently. For example, a mobile phone component has a lower application risk than a security system component of a nuclear power plant. Thus, five levels are distinguished by increasing risks: I to V.

A case study using the SCETM was developed in order to provide the first insights of its usability and viability to evaluate software components quality using those techniques presented. Level 1 and Level 2 from SCETM were used during the study. More information about it can be found at (Alvaro et al., 2007b).

Besides, the proposed paradigm used to track the properties of the whole component quality framework was presented. Some few examples of usage were provided in order to show the GQM applicability and usage. With the idea of helping the evaluation team during its first software component evaluation, a set of valuable metrics examples are presented in Appendix A.



Component Evaluation Process

The definition of the Component Quality Model (CQM), the Evaluation Techniques Framework (through the Software Component Techniques Model (SCETM)) and the Metrics Framework presented, in previous chapters, were important to introduce the quality attributes that must be considered to software component, to evaluate those quality attributes using some pre-defined and well-established techniques, and, to measure those attributes through the GQM paradigm. However, it is essential to define a set of activities that should be followed in order to guide the evaluation team during the component evaluation. In this way, the component evaluation could be repeatable and reproducible once each activity contains a well-detailed description, its *inputs* and *outputs*, *mechanisms* to execute and to *control*.

A consistent and good evaluation results can only be achieved by following a high quality and consistent evaluation process (Comella-Dorda et al., 2003). This does not mean that each evaluation activity requires a highly complex, exquisitely documented process (although sometimes they do), but if you do not follow some kind of consistent process, it is likely that the quality of your results will vary.

In this sense, a Component Evaluation Process was proposed (Alvaro et al., 2007c), (Alvaro et al., 2007d) in order to define more precisely the activities necessary to evaluate the component quality.

7.1 The Component Evaluation Process

As presented in chapter 3, the Component Evaluation Process is based on the SQuaRE project, which provides guidance and requirements for the software product evaluation. The idea is to follow as much as possible this standard to

develop a new one for software component quality evaluation. Moreover, a set of works from literature which includes processes for software product evaluation and processes for software component assessment aided during the definition of this process (McCall et al., 1977) , (Boegh et al., 1993), (Beus-Dukic et al., 2003), (Comella-Dorda et al., 2002).

In this context, a set of activities to guide the evaluation team during the evaluation was proposed (Figure 7.1), which is presented using SADT notation (Ross, 1997).

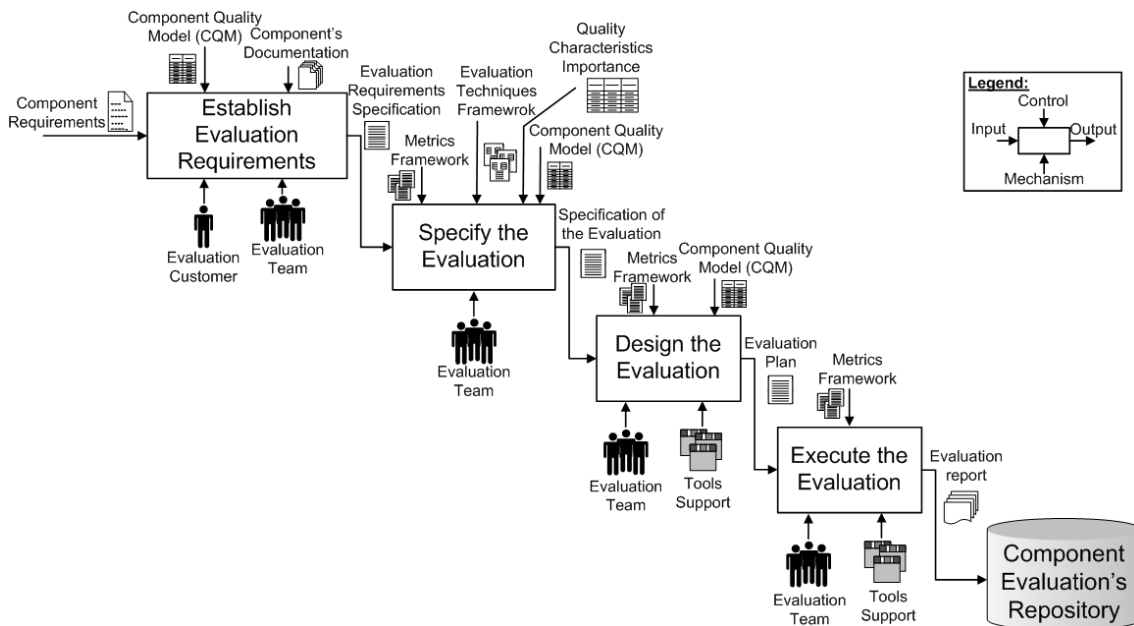


Figure 7.1. Component Evaluation Process.

Figure 7.1 has shown the macro-view of the whole component evaluation process and Appendix B presents a template to guide the evaluation team during the whole component evaluation process. Now, each activity of the process will be described in more details.

7.1.1. Establish Evaluation Requirements activity

This activity includes specifying the goals and scope of the evaluation, and specifying evaluation requirements. The evaluation requirements specification should identify the quality characteristics (using the *Component Quality Model (CQM)*), but also other aspects such as stakeholders to compose the evaluation team, the component's constraints and the component relationships with the software system or with the target environment. Moreover, when defining the evaluation team, the stakeholders should be carefully selected in order to assure a high-quality component evaluation.

The inputs for this activity are the *Component Requirements Document*, the *Component's Documentation* available and the *Component Quality Model (CQM)*. Based on these inputs, the evaluation team together with the evaluation customer will *Establish the Evaluation Requirements* and generate the *Evaluation Requirements Specification* as output, as shown in Figure 7.2.

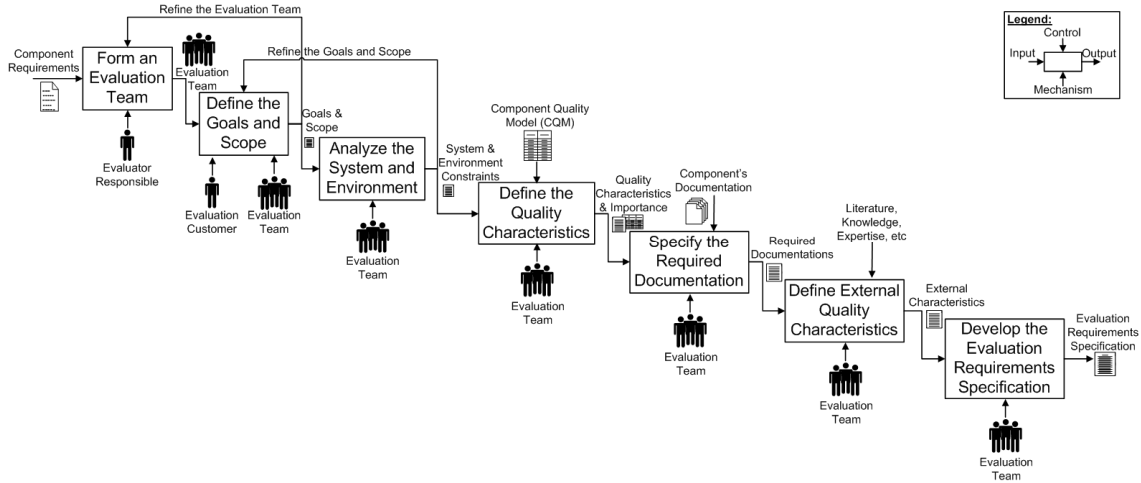


Figure 7.2. Establish Evaluation Requirements steps.

Figure 7.2 has shown the steps that should be followed execute the *Establish Evaluation Requirements* activity which will be presented next.

- Form an Evaluation Team:** The importance of an effective evaluation team should not be underestimated, since without an effective team, an evaluation cannot succeed. People with different skills are essential to the evaluation team, such as: technical experts, domain experts, business analysts, contract personnel, end users, among others. A good balance of power is also important for a good team. The other skills necessary will depend on the evaluation scope and objectives, and can include security professionals, maintenance staff, etc.

According to (Comella-Dorda et al., 2003), to be successful the evaluators need to: **(i)** understand the impact of the software component on the system development process and on the target environment, **(ii)** determine evaluation requirements for software component; **(iii)** define software component quality attributes; **(iv)** select software component evaluation techniques; and **(v)** employ a software component evaluation process that address the inherent tradeoffs.

While there are no rules for identifying evaluation stakeholders, errors of inclusion (i.e. including additional individuals or groups) are less risky than errors of exclusion (i.e. omitting legitimate stakeholders). Thus, if there is any doubt about including or not one stakeholder, it is more interesting to add him/her to the evaluation process. However, the number of stakeholders that will compose the evaluation team will depend of several aspects, such as: the size of the component, the complexity of the problem to be solved, the target domain, the algorithm complexity, among other factors. The essential stakeholders in any kind of evaluation are the customer and the evaluator responsible for it (which has large knowledge, at least, in the target domain and in the component technology). More stakeholders could be added to the evaluation team, however, according to (Comella-Dorda et al., 2003), more than 7 stakeholders are not so interesting to the evaluation;

- **Define the Goals and Scope:** Some important questions should be answered before the evaluation tasks begin, such as:
 - What is the evaluation expected to achieve?
 - What are the responsibilities of each member of the team?
 - When should the evaluation finish?
 - What constraints must the evaluation team adhere to?
 - What is the related risk of the component to its target domain?

Often, this basic information is not documented. Moreover, the people tend to be more productive when they have a clear picture of the ultimate goals of the project, in this case, the component evaluation process. Thus, the evaluation team should answer these questions, focusing on:

- Goals of the evaluation;
- Scope of the evaluation;
- Names of team members and their roles;
- Component and domain risk-level;
- Statement of commitment from both stakeholder(s) and customer(s); and

- Summary of decisions that have already been made.

So, the evaluation team defines the goals, the scope and the related risk-levels of the component domain together with the evaluation customer.

Moreover, it is important to describe, in a general way, the component functionalities and the domain of the component in order to understand them more precisely;

- **Analyze the System and Environment:** The evaluation team should analyze the software system in order to define the impact of the software component in these system(s), which requirements and functionalities have impact, the architecture constraints, programming language constraints, environment constraints, etc. On the other hand, if the component will be evaluated independent of a software system, the evaluation team should define, as precise and detailed as possible, the environment that the component will be evaluate (i.e. target deployment environment, target domains of this component, version of the related tools, supplier of those tools, environment constraints, etc.). This step is important to define how big will be the complexity of the component's evaluation using that environments or that system specified, answering a set of questions:
 - How much effort will be spent to provide the whole infrastructure to evaluate the component? How are the complexity and constraints of this(ese) environment(s)?
 - What is the size of those selected system? What is(are) the target domain(s) of those systems?
 - What is the impact of the software component into selected system?
 - What are the component dependencies?

Related to these last questions, the evaluation team should analyze if the component has dependencies with other components, modules, systems, etc. So, all dependencies should be described in this step in order to comprehend the behavior of the software components (dependencies, cohesion, coupling, etc.);

- **Define the Quality Characteristics:** This step will define a set of quality characteristics that should be considered during the component's evaluation. Based on the Component Quality Model (CQM), the evaluation team must define the quality characteristics and the sub-characteristics that will be used to evaluate the software component. It is interesting that the importance related to each quality characteristic should be defined. Thus, this identification will aid the next activity where the team evaluator should define the depth of the evaluation. Next, the evaluation team should discuss the selected quality characteristics and their related importance with the customer in order to achieve an agreement between both sides. The importance levels of the component could be: **1**-Not Important; **2**-Indifferent; **3**-Reasonable; **4**-Important; **5**-Very Important. A simple table can help their in this identification, as show in Table 7.1.

Table 7.1. Example of Importance's definition.

Characteristics	Sub-Characteristics	Importance
Functionality	Accuracy	4
Functionality	Security	3
...

- **Specify the Required Documentation:** Based on the quality characteristics selected and their importance level, the evaluation team should define which documentation, assets and information are necessary to execute the evaluation. This information should be shared with the customer in order to him/her provide the information required. After obtaining those documents and attached them into the evaluation process, the next step could be performed;
- **Define External Quality Characteristics:** Besides the quality characteristics presented in the CQM, it is possible that exists quality characteristics that are not covered by that model. In this case, the evaluation team should define the “new” characteristic and reference it based on considerable works in the literature in order to clarify the definition and the importance of the quality characteristic. Still on, the evaluation team should define the sub-characteristics and quality attributes of this “new” characteristic defined and attach all

information in the Table 7.1 presented later, and, at the end of the component evaluation process the team should analyze if it is interesting to put those external quality characteristics in the CQM;

- **Develop the Evaluation Requirements Specification:** The last step of this activity is to elaborate a document with all information collected during the *Establish Evaluation Requirements* activity and generate the *Evaluation Requirements Specification*, which is the main input of the next activity.

7.1.2 Specify the Evaluation activity

This activity includes defining the evaluation level, through *the Guidelines for Selecting Evaluation Level*; the definition of the evaluation techniques to be used to evaluate each level defined previously, through the *Evaluation Techniques framework*; and selecting the metrics that will be used to collect information about all steps of the evaluation process, through the *Metrics Framework*. The goal here is to detail as much as possible the specification level in order to assure the reproducibility and repeatability of the evaluation. The idea is to assure that other groups of people, which does not participate in the evaluation, can understand and execute the same evaluation again.

The evaluation team should define metrics to track the properties of the quality characteristics and the techniques to be adopted as well as the whole evaluation process. An important aspect here is to consider the complexity of obtaining the data of each metric required and if the selected metric can represent completely the information required by the evaluation team.

The inputs of this activity is the *Evaluation Requirements Specification*. The *Quality Characteristics Importance*, defined in the last activity, acts as control in this activity. Thus, using those assets, the evaluation team will develop the *Evaluation Requirements Specification* and generate the *Specification of the Evaluation* as output, as shown in Figure 7.3.

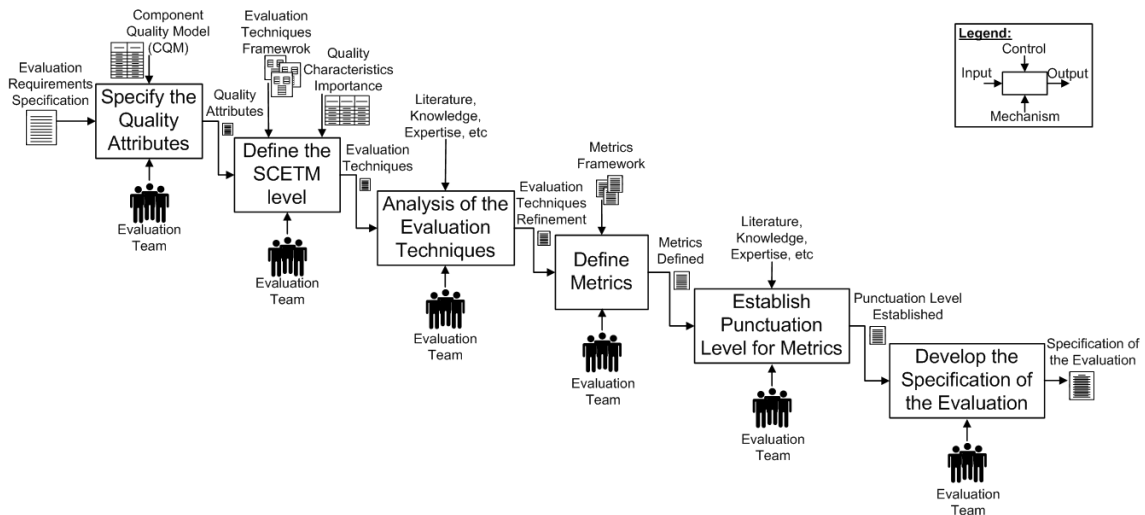


Figure 7.3. Specify the Evaluation steps.

Figure 7.3 has shown the steps that should be followed to execute the *Specify the Evaluation* activity which will be presented next.

- Specify the Quality Attributes:** Based on the characteristics and sub-characteristics defined in the previous activity (*Define the Quality Characteristics* step and *Define External Quality Characteristics* step), the evaluation team needs to define the quality attributes of each sub-characteristics. For those CQM's quality characteristics, the evaluation team can use the CQM in order to help them in the definition of the quality attributes. And, for those that are external characteristics, the evaluation team should define the quality attributes for composing and complementing the quality characteristic (if it was not performed yet).

Thus, the evaluation team assures the complete definitions of the quality characteristics necessary to evaluate the software components, as shown in Table 7.2. After this step, no more characteristics will be added to the evaluation process and, thus, the evaluation team must guarantee the complete definition of those quality characteristics, as much as possible.

Table 7.2. Example of Quality Attributes Definition.

Characteristics	Sub-Characteristics	Quality Attributes	Importance
Functionality	Accuracy	Correctness	4
Functionality	Security	Data Encryption	3
...

- Define the Software Component Techniques Model (SCETM) level:** After specifying all quality characteristics and attributes, the evaluation team must define which evaluation technique should be used to evaluate those attributes. The evaluation team should consider the importance of each quality characteristic (developed in the *Define the Quality Characteristic* step), described in the last activity. Thus, they will define the SCETM level that the software component should attend, using a set of guidelines for selecting evaluation level as basis for this decision (presented in chapter 6 in Table 6.1). Moreover, the evaluation team has two directions to follow: **(i)** they can define a unique level to evaluate all quality attributes of the software component using the techniques of this level (e.g. SCETM II level) or, **(ii)** if necessary, they can select a mix of levels, where each quality characteristic will be evaluated in a certain SCETM level (e.g. functionality will be evaluate on SCETM I level, reliability will be evaluated on SCETM III level and so on). This decision will affect the final report and should be discussed with the customer. In other words, if the evaluation team decides the first direction, at the end, the component will be certified or not for a certain level (e.g. certified in SCETM II level or not). And, if they choose a mix of levels, at the end, the report will contain the level achieved by each characteristic (e.g. Functionality – SCETM II, Reliability – SCETM I, Usability – SCETM III, etc). For this reason, the decision of the directions to be followed during the evaluation should be communicated to the customer before going to the next step. Table 7.3 shows an example of the decision of the evaluation techniques for each quality attributes.

Table 7.3. Example of Define SCETM.

Characteristics	Sub-Characteristics	Quality Attributes	Importance	SCETM Level / Evaluation Technique
Functionality	Accuracy	Correctness	4	II. Black-Box Testing
Functionality	Security	Data Encryption	3	III. Code Inspection
...

Still on, if any external quality characteristic was proposed in last activity (developed in *Define External Quality Characteristics* step), the evaluation team should define how this “new” quality characteristic is to be evaluated through techniques available on the literature or through one of the techniques represented in the Techniques Evaluation framework (presented in chapter 6);

- **Analysis of the Evaluation Techniques:** The evaluation team, using their expertise, knowledge and know-how in the evaluation techniques, must analyze these and decides if those techniques are useful to evaluate the target software component or if it is necessary to define other techniques for executing / complementing the evaluation. The idea is to define the best technique for each kind of evaluation. It is important to consider here that all of those techniques presented in SCETM are recommended techniques to guide the evaluation team during the selection process. However, it is not avoided the proposal of new techniques that are better to evaluate the target software component. Moreover, the evaluation team should justify the adoption of a new technique and if it is reasonable, it can be incorporated to the SCETM;
- **Define Goal-Question-Metric (GQM):** The evaluation team will define metrics to track the properties of the quality characteristics selected, the techniques adopted, as well as the whole evaluation process. Through the Metrics Framework (described in chapter 6) the evaluation team will define:
 - the metrics to evaluate those quality attributes selected from CQM or from external sources;
 - the metrics necessary for each SCETM technique used(which is necessary at least one metric for each evaluation technique that will be used); and
 - the metrics to measure the efficiency of the component evaluation process;

The information collected using these metrics will support the quality assurance of the component evaluation and also provides insights for

the next evaluations once they will be available on the *Component Evaluation's Repository*;

- **Establish Score Level for Metrics:** After defining all the metrics, the evaluation team should consider a score level to facilitate its analysis. For example, in a certain component evaluation if the metric defined for measuring the *Suitability* quality attribute achieved less than 40%, it is not accepted; between 41% and 80% could be considered reasonable; and higher than 81% it is considered acceptable and could receive the evaluation agreement. This score level will be dependent on the importance of each quality characteristic (*Define the Quality Characteristic* step) and the evaluation level (*Define SCETM level* step) defined during this activity. However, there is a kind of scale that should be considered as basis in order to start the establishment of the score level. The scale is defined on ISO/IEC 15504-2 (ISO/IEC 15504-2, 2000) (the ISO/IEC 25000 does not contain any kind of scale for quality achievement level and, in this way, another standard guiding this thesis in the definition of the range of those scales), as follows:

- **N (Not Achieved):** 0% to 15% - There is no or little evidence about the presence of the quality attribute on the component;
- **P (Partially achieved):** 16% to 50% - There is evidence about the presence of the quality attribute on the component. However, the quality attribute aspects is partially achieved;
- **L (Largely achieved):** 51% to 85% - There is evidence about the presence of the quality attribute on the component. The component provide the quality attribute aspects implemented but it is not completely achieved; and
- **F (Fully achieved):** 86% to 100% - The component provide all necessary fulfillments for the quality attribute under evaluation.

The evaluation team should analyze if the scale proposed on ISO 15504-2 makes sense to the component in evaluation. This is dependent on the reliability level expected by the component, i.e. the SCETM level defined to evaluate the component. If needed, the

evaluation team must do some improvements in the scale proposed in order to become more accurate to the component under evaluation.

Moreover, the evaluation team should retrieve the previous evaluation in the *Component Evaluation's Repository* in order to analyze the score level for the metrics defined in previously evaluations;

- **Develop the Specification of the Evaluation:** The last step of this activity is elaborating a document with all information collected during the *Specify the Evaluation* activity and generate the *Specification of the Evaluation*, which is the main input for the next activity.

7.1.3 Design the Evaluation activity

This activity needs to consider access to component documentation, development tools and personnel, evaluation costs and expertise required, the evaluation schedule and costs, the description of the evaluation environment as detailed as possible, and, reporting methods and evaluation tools.

The input for this activity is the *Specification of the Evaluation*. The *GQM*, defined in the last activity, acts as control in this activity. Based on these inputs, the evaluation team, using a set of tools as support, will develop the *Evaluation Plan*, which contain detailed and complete information about the evaluation process that should be follow, as show in Figure 7.4.

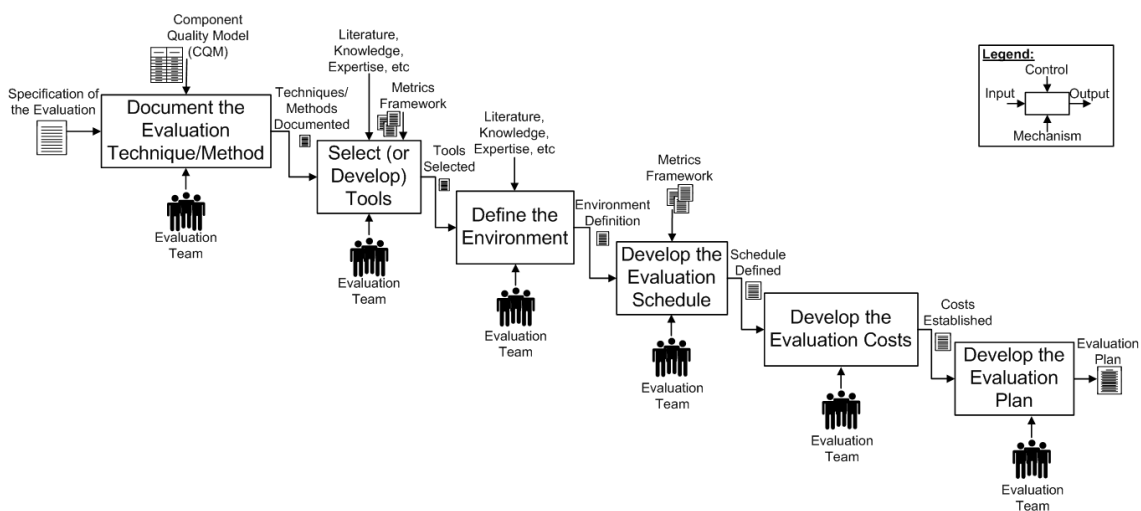


Figure 7.4. Design the Evaluation steps.

Figure 7.4 has shown the steps that should be followed to execute the *Design the Evaluation* activity which will be presented next.

- **Document the Evaluation Technique/Method:** The objective of this step is to document all those evaluation techniques or methods proposed to evaluate the component in the last activity (*Define the Software Component Techniques Model (SCETM) level* step). The idea is to describe them in order to all stakeholders involved in the evaluation process understand the technique/method and can execute the tasks to evaluate the software component.

All those techniques proposed on SCETM should have its description stored in the *Component Evaluation's Repository* and should be reused in order to increase the productivity in this step. However, if a new technique was adopted in the last activity (*Define SCETM level* step), it is necessary develop this documentation from scratch and, at least, should contain: the name of the technique/method, the reference in literature for this technique, the description and, if necessary, the use guide. The depth of this description will depend on the knowledge in that technique/method that the stakeholders involved in the evaluation team have;

- **Select (or Develop) Tools:** Based on a set of tools available in literature, on the programming language and on the environment that the component should be deployed (or system in which it should be evaluated), the evaluation team needs to select the tools. Thus, the evaluation team will select the tools that support the execution of the techniques selected previously (*Define the Software Component Techniques Model (SCETM) level* step). If necessary, it is possible to develop a specific tool that evaluates a certain (or a set of) technique(s) in the case that the cost/benefit to develop one tool is justified. Still on, the evaluation team defines if it is necessary any tool to collect the metrics defined in later activity (*Define GQM* step). Table 7.4 shows an example of a Table that could be used to store the selected tools for each quality attribute.

The description, at least, should consist of: the name of the tool, the version of the tool, the origin of the toll (i.e. its supplier) and the description of the technique (as detailed as the evaluation team consider necessary);

Table 7.4. Example of Tools Selection.

Characteristics	Sub-Characteristics	Quality Attributes	SCETM Level / Evaluation Technique	Tool used
Functionality	Accuracy	Correctness	II. Black-Box Testing	Junit ¹⁵ , FindBugs ¹⁶
Functionality	Security	Data Encryption	III. Code Inspection	PMD ¹⁷
...

- **Define the Environment:** Now, it is time to describe the environment that the component will be evaluated in order to establish the context of the evaluation. The main input for this step is the *Analyze the System and Environment* step developed during the first activity. Thus, the evaluation team will specify the whole environment as detailed as possible (i.e. software and tools necessary, the versions of the used software/tools, environment installed those software/tools, software/tools and environment constraints, etc.). Based on the environment defined, the evaluation team will analyze the component quality. It is important to remind that the final report will describe the environment(s) under which the component was evaluated;
- **Develop the Evaluation Schedule:** After all technological steps were developed, now it is time to analyze all available resources, such as tools, stakeholders, techniques, methods, etc. So, the evaluation team should develop the evaluation schedule with activities and tasks for each stakeholder and the time to execute each task. It is interesting to achieve an agreement with all stakeholders involved in the evaluation process in order to minimize the risks during the component evaluation;
- **Develop the Evaluation Cost:** The establishment of costs of a software project or an evaluation project has been a hard task.

¹⁵ <http://www.junit.org/>¹⁶ <http://findbugs.sourceforge.net/>¹⁷ <http://pmd.sourceforge.net/>

However, in order to define the evaluation costs a few and simple guidelines can be proposed, as following:

- Number of stakeholders involved in the evaluation;
- Skills and experience of those stakeholders;
- Definition of the tasks for each stakeholder;
- Time defined to each stakeholder to execute his/her tasks;

Based on these guidelines, the evaluator responsible can determine the cost of each stakeholder involved in an evaluation process through the *stakeholder skills X task to execute X time define to execute each task*, and, thus, calculate the total costs of the evaluation process.

The idea is to define the costs of the evaluation in order to the customer knows the investment in the component evaluation and approves it. This is not the best approach to define the costs; however, this is the first step towards to the costs definition. Thus, the intention is to acquire expertise during a set of initial evaluations, store these in the *Component Evaluation's Repository* and, provide these data to the next component evaluation in order to have some data to be compared and to refine the costs of the whole process;

- **Develop the Evaluation Plan:** The last step of this activity is to elaborate a document with all information collected during the *Design Evaluation* activity. The output of this activity is the *Evaluation Plan*, which is the main input for the next activity.

7.1.4 Execute the Evaluation activity

This activity includes the execution of the evaluation and analysis of the evaluation results. The conclusions should state whether the software component is appropriate for use in the intended environment (and maybe in system(s)) described in later activities.

The input of this activity is the *Evaluation Plan*. The *GQM*, defined in the second activity, acts as control in this activity. Based on these input, the evaluation team, using a set of support tools, will develop the *Evaluation Report* to delivery to the customer and to be stored in the *Component Evaluation's Repository* too, as shown in Figure 7.5.

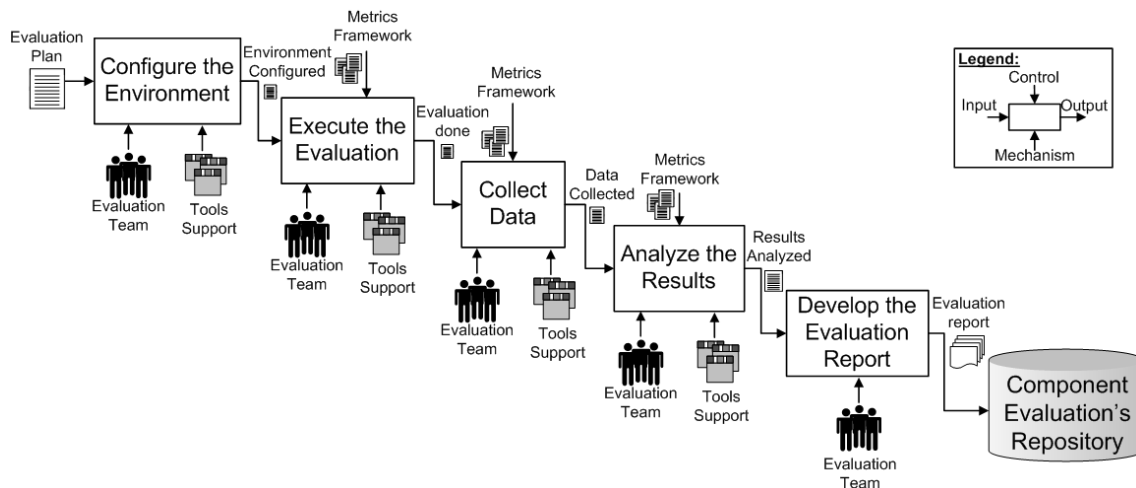


Figure 7.5. Execute the Evaluation steps.

Figure 7.5 has shown the steps that should be followed to execute the *Execute the Evaluation* activity which will be presented next.

- **Configure the environment:** Based on *Environment*, defined in last activity, the stakeholder(s) responsible for this activity will configure the environment in order to start the evaluation of the component;
- **Execute the Evaluation:** Based on the *Evaluation Plan*, the stakeholder(s) responsible for this activity will apply the techniques/methods, using the defined tools and the metrics adopted to collect information about the component evaluation;
- **Collect Data:** Collecting data provides a basis for analysis. Good data collection is simple, repeatable, measures what is intended to measure, and captures information in a form suitable for analysis. Accurate data collection is one of the keys to successful software component evaluation, yet the act of collecting data is full of surprises – a few good ones and more than a few bad ones (Comella-Dorda et al., 2003).

It is important to remember that the quality of the evaluation is only as good as the data collected. Confidence in the final results can be improved by ensuring that the data collection is as accurate as possible.

Thus, the collection process should be carefully executed in order to provide good data to the next step. All data obtained from the tools used must be stored in a table like Table 7.5, based on the metrics and

interpretation defined in *GQM Define* step for each quality attribute. So, the data analysis will be executed based on this table provided by this step;

Table 7.5. Example of Data Collection.

Characteristics	Sub-Characteristics	Quality Attributes	SCETM Level / Evaluation Technique	Tool used	Results	Metric
Functionality	Accuracy	Correctness	II. Black-Box Testing	Junit, FindBugs	0.7	Precision on results / Amount of tests
Functionality	Security	Data Encryption	III. Code Inspection	PMD	0.8	N° interface encrypted / number of services
...

- **Analyze the Results:** After finishing the execution, the evaluation team will pick up all data stored in the previous table to analyze and develop a report about the evaluation. Those data should be carefully analyzed and correlated each other in order to analyze the component quality in a complete view of its functionalities. It means that one data result can affect/interfere in other data result and vice-versa, becoming important to correlate the results.

If possible, it is interesting compare this evaluation with other similar ones stored in *Component Evaluation's Repository* in order to provide insights to the evaluation team during the data analysis. After that, this report will be stored in a *Component Evaluation's Repository* for sharing the knowledge acquired during this evaluation to the next component evaluations;

- **Develop the Evaluation Report:** The final result is an *Evaluation Report* that describes all quality attributes chosen, all techniques defined, the methods, processes and tools used, the metrics defined for evaluation, the data collected in the last step and the evaluation results.

If possible, the evaluation team can describe a set of suggestions in order to improve the quality of the component. The goal of the

recommendation is to provide some information so that the customer can improve the component quality. The evaluators learn many lessons about the use of the component during its evaluation (i.e. component architecture, deployment constraints, tailoring, wrapping, and testing and maintenance activities) and can contribute with important recommendations to the customer. This information is important in the case that the component is approved and, much more interesting if the component is rejected.

7.1.5 Process Summary

Some individuals believe that following any documented process is a waste, particularly when the end goal is to save time and money. According to (Comella-Dorda et al., 2003), the highly informal COTS evaluation processes share the blame for the failure due to their lack of activities to follow. The process described here is a means of performing component evaluations and not an end in itself. Expect to tailor this process for specific purpose, and do not let it get in the way of getting good data and making an informed recommendation.

Finally, it is important to consider that the evaluation team is the main responsible for execute this process and should be carefully defined in order to assure that the evaluation will be efficiently developed.

7.2 Summary

This chapter presented the Component Evaluation Process, which is composed of a set of activities to guide the evaluation team during the component evaluation task. Each activity contains a set of steps that should be followed and its details were carefully described. A template was defined in order to guide the team evaluation during the whole process and is presented in Appendix B.

Time, effort and human resources needed for applying this process together with the whole framework always depend on the importance, size and the component risk-level. Moreover, the complexity of the techniques selected to evaluate the component is a factor that could increase considerably the time spent on the component evaluation. Thus, the component evaluation in general may range from one evaluator assure quality for 1 day, to a pool of evaluators for a month.



Experimental Study

The task of choosing the best software engineering techniques, methods, processes and tools to achieve a set of quality goals under a given scenario is not a simple endeavor. Experimental studies are fundamental for executing cost-benefit analysis of software engineering approaches. Based on empirical evidences, one can construct experience bases that provide qualitative and quantitative data about the advantages and disadvantages of using these software engineering techniques, methods, processes and tools, in different sets and domains. According to Basili et al. (Basili et al., 1996b) experimentation in software engineering is necessary because hypotheses without proofs are neither safe nor reliable as a knowledge source. Moreover, replication is an important aspect in this scenario.

According to Fusario et al. (Fusario et al., 1997), replicate means to reproduce as faithfully as possible a previous experiment, usually run by other researchers, in a different environment and conditions. When the results generated by a replication are coincident with the ones of the original experiment, they contribute to strengthen the hypotheses being studied. Otherwise, other parameters and variables should be investigated.

In this way, new software engineering techniques, methods, processes and tools must be experimented in order to consider how and when they really work; to identify their limits; and to understand how to improve them. Thus, in order to determine whether the Software Component Quality Framework meets its proposed goals, an empirical study should be planned. This chapter describes the definition, planning, operation, analysis and interpretation of the experimental study.

8.1 Software Component Quality Framework: An Experimental Study

According to Wohlin et al. (Wohlin et al., 2000), the experimental process can be divided into the following main activities: the **definition** is the first step, where the experiment is defined in terms of problem, objective and goals. The **planning** comes next, where the design of the experiment is determined, the instrumentation is considered and the threats to the experiment are evaluated. The **operation** of the experiment follows from the design. In the operational phase, measures are collected, analyzed and evaluated in the **analysis** and **interpretation**, providing some conclusions to the experiment. Finally, the results are **presented** and **packaged**.

The plan of the experimental study to be discussed follows the model proposed in (Wohlin et al., 2000) and the organization adopted in (Barros et al., 2002), as presented next. The definition and planning activities will be described in future tense, showing the logic sequence between the planning and operation.

8.2 Definition of the Experimental Study

According to the Goal Question Metric Paradigm (GQM) (Basili et al., 1986), the main objective of this study is to:

*Analyze the software components quality framework
for the purpose of evaluating
with respect to the feasibility and its usage
from the point of view of the researchers and quality engineers
in the context of a domain engineering project.*

8.3 Planning of the Experimental Study

Context. The objective of the study is to evaluate the feasibility of the Software Component Quality Framework proposed. It is based on a set of software components developed during a domain engineering project accomplished in a university lab. The requirements of the project were defined by the experimental staff based on real-world projects. During the domain engineering project the subjects start to apply and collect useful data to evaluate the software components quality at the end of the project. The study will be

conducted as single object of study which is characterized as being a study which examines an object on a single team and a single project (Basili et al., 1986).

Training. The training of the subjects using the process will be conducted in a classroom at the university. The training will be divided in two steps: in the first one, concepts related to software reuse, component-based development, variability, domain engineering, software product lines, asset repository, software reuse metrics, software reuse processes, software component quality, software component evaluation, testing and inspection will be explained during eleven lectures with two hours each. Next, the domain engineering process and software component quality framework will be discussed during four lectures (two for each). During the training, the subjects can interrupt to ask issues related to lectures. Moreover, the training will be composed of a set of slides and recommended readings.

Pilot Project. Before performing the study, a pilot project will be conducted with the same structure defined in this planning. The pilot project will be performed by a single subject, which is the author of the proposed framework. For the project, the subjects will use the same material described in this planning (which is developed by the author of the framework during the pilot project), and will be observed by the responsible researcher. In this way, the pilot project will be a study based on observation, aiming to detect problems and improve the planned material before its use.

Selection of Subjects. All the students that registered in the Software Engineering pos-graduate course at Federal University of Pernambuco, Brazil, were selected (twelve students). In this way, the subjects were selected by convenience sampling (Wohlin et al., 2000) representing a non-random subset from the universe of students from Software Engineering. In convenience sampling, the nearest and most convenient people are selected as subjects.

Subjects. From twelve students, three of them with more experience in software quality were selected to act in this experiment. Thus, the subjects of the study (three subjects), according to their skills and technical knowledge, will act as evaluation leader, architecture/environment specialist and programming language specialist.

Instrumentation. All the subjects will receive a questionnaire (Appendix C) about his/her education and experience, besides the subjects received the chapters 5, 6 and 7 of this Thesis which contain the software component quality framework. At the end of the experimentation, the intention is to provide a questionnaire for the evaluation of the subjects' satisfaction using the framework.

Criteria. The quality focus of the study demanded criteria that evaluate the real feasibility of the framework in measuring software components quality and the difficulty of the framework usage. This criteria will be evaluated quantitatively ((i) coverage of the CQM; (ii) coverage of the SCETM; and (iii) subjects' difficulty to use the framework).

Hypotheses. An important aspect of an experimental study is to know and to formally state what is going to be evaluated in the experimental study. Hence, a set of hypotheses was selected, as described next.

- **Null hypotheses, H_0 :** these are the hypotheses that the experimenter wants to reject strongly. The following hypotheses can be defined, using GQM:

Goal. To determine the feasibility of the framework to measure the software component quality and to evaluate the difficulties to use the framework.

Question.

1. Does the quality attributes proposed on CQM is used during the component evaluation?
2. Does the evaluation techniques proposed on SCETM is used during the component evaluation?
3. Do the subjects have difficulties to apply the framework?

Metric.

H_0' : coverage of the component quality attributes proposed in the CQM X the quality attributes used during the component evaluation < 85%

H_0'' : coverage of the evaluation techniques proposed on the SCETM for the quality attributes defined during the component evaluation < 85%

H_0''' : %Subjects that had difficulty to understand, follow and use the Software Component Quality Framework > 20%

The CQM proposed must contain the major quality attributes necessary to any kind of software component evaluation. In this sense, the null hypothesis

H_0 states that the coverage of the component quality attributes proposed in the CQM X the quality attributes used during the component evaluation is less than 85%. It may exist some components in specific kind of domains that need new quality attribute(s) in order to measure specific characteristic of the component.

After that, following the component evaluation process, the evaluation team should define the techniques that will be used to evaluate each quality attribute proposed previously. In this way, the null hypothesis H_0 states that the coverage of the evaluation techniques proposed on the SCETM for the quality attributes defined on the component evaluation is less than 85%.

At least, the component evaluation framework is consisted of four modules and there is a set of steps to follow in order to accomplish the component evaluation. In this way, the null hypothesis H_0 states that the subjects that will have difficulty to understand, follow, and use the whole software component quality framework is more than 20%.

The values of these hypotheses (85%, 85% and 20%, receptivity) were achieved through the feedback of some researchers of RiSE group and, software and quality engineers of a Brazilian software company CMMi level 3. Thus, these values constitute the first step towards well-defined indices which the framework must achieve in order to indicate its viability.

- **Alternative hypotheses:** these are the hypotheses in favor of that which the null hypotheses reject. The experimental study aims to prove the alternative hypotheses by contradicting the null hypotheses. According to the selected criteria, the following hypotheses can be defined:

Goal. To determine the feasibility of the framework to measure the software component quality and to evaluate the difficulties to use the framework.

Question.

1. Does the quality attributes proposed on CQM is used during the component evaluation?
2. Does the evaluation techniques proposed on SCTEM is used during the component evaluation?
3. Do the subjects have difficulties to apply the framework?

Metric.

H₁: *coverage of the component quality attributes proposed in the CQM X the quality attributes used during the component evaluation $\geq 85\%$*

H₂: *coverage of the evaluation techniques proposed on the SCETM for the quality attributes defined on the component evaluation $\geq 85\%$*

H₃: *%Subjects that had difficulty to understand, follow and use the Software Component Quality Framework $\leq 20\%$*

Independent Variables. The independent variables are the education and the experience of the subjects, which will be collected through the questionnaire and the proposed framework. This information can be used in the analysis for the formation of blocks.

Dependent Variable. The dependent variables are the quality of the CQM and SCETM developed and the usability of the framework proposed to assure the component quality. The quality of the CQM and SCETM will be measured through its feasibility. And the quality of the framework will be measured through the capacity of the “users” understand, use and execute in a correctly way all the steps of the framework.

Qualitative Analysis. The qualitative analysis aims to evaluate the difficulty of the application of the proposed framework and the quality of the material used in the study. This analysis will be performed through a questionnaire (Appendix C). This questionnaire is very important because it will allow evaluating the difficulties that the subjects have with the proposed models and, consecutively, with the whole framework, evaluating the provided material and the training material, and improving these documents in order to replicate the experiment in a near future. Moreover, this evaluation is important because it can verify if the material is influencing the results of the study.

Randomization. This technique can be used in the selection of the subjects. Ideally, the subjects must be selected randomly from a set of candidates. However, as cited in the Selection of Subjects section, the subjects were selected by convenience sampling.

Blocking. Sometimes, there is a factor that probably has an effect on the response, but the experimenter is not interested in that effect. If the effect on the factor is known and controllable, is possible to use a design technique called blocking. Blocking is used to systematically eliminate the undesired effect in the

comparison among the treatments. In this study, it was not identified the necessity of dividing the subjects into blocks, since the study will evaluate just three factors, which are the completeness of CQM and SCETM, and the framework usage.

Balancing. In some experiments, balancing is desirable because it both simplifies and strengthens the statistical analysis of the data. However, in this study it is not necessary to divide the subjects, since there is only one group.

Internal Validity. The internal validity of the study is defined as the capacity of a new study to repeat the behavior of the current study, with the same subjects' expertise and objects with which it was executed (Wohlin et al., 2000). The internal validity of the study is dependent of the number of subjects. This study is supposed to have at least between two to five subjects to guarantee a good internal validity.

External Validity. The external validity of the study measures its capability to be affected by the generalization, i.e., the capability to repeat the same study in other research groups (Wohlin et al., 2000). In this study, a possible problem related to the external validity is: **(i)** the subjects' motivation, since some subjects can perform the study without responsibility or without a real interest in performing the project with a good quality as it could happen in an industrial project; and **(ii)** the subjects' experience, once the background and experience in software area (including software development, tests and quality area) could be lower than the expected in this experiment. The external validity of the study is considered sufficient, since it aims to evaluate the viability of the application of the software component quality framework. Since the viability is shown, new studies can be planned in order to refine and improve the process.

Construct Validity. The validation of the construction of the study refers to the relation between the theory that is to be proved and the instruments and subjects of the study (Wohlin et al., 2000). In this study, a relative know project will be used (i.e. the subjects have about one and a half years of experience in this kind of project). Thus, this choice avoids previous experience of making a wrong interpretation of the impact of the proposed framework.

Validity of the Conclusion of the Study. The validation of the conclusion of the study measures the relation between the treatment and the result, and determines the capability of the study to generate conclusions (Wohlin et al., 2000). This conclusion will be drawn by the use of descriptive statistic.

8.4 The project used in the Experimental Study

The project used in the experimental study was to perform the domain engineering of a set of tools developed by RiSE group during the last four years. The idea is to use the RiSE Domain Engineering process (RiDE) (Almeida, 2007) to execute the domain analysis, domain design and domain implementation in order to analyze the commonalities and variability between the tools that RiSE is developing, and guiding also the development of further tools. Moreover, during the usage of the RIDE process, the subjects will define the stakeholders responsible to execute the component evaluation quality and will evaluate the software components produced by RIDE process using the Software Component Quality Framework proposed in this thesis.

At the end of the project, the subjects will perform the domain engineering of those tools and evaluate the quality of the software components produced, which is the focus of this experiment.

8.5 The Instrumentation

Instrumentation. Before the experiment can be executed, all experiment instruments must be ready. These include the experiment objects, guidelines, forms and tools. In this study, the questionnaire presented on Appendix B and Appendix C, in conjunction with the papers about the process were used. The questionnaires presented the subjects' names in order to check additional information or misunderstanding. However, the subjects were notified for the information confidentially.

8.6 The Operation

Experimental Environment. The experimental study was conducted during part of a M.Sc. and Ph.D. Course in Software Reuse, during November 2007 – June 2008, at Federal University of Pernambuco. The experiment was

composed of three subjects and all the evaluation process was developed in 135 hours and 48 minutes (Figure 8.1 shows the time spent in each activity of the evaluation process). The evaluation process activity was conducted using five components (described next) generated from the usage of the RiDE process with the idea of developing a simple asset search/retrieval tool.

- **Persistence Manager:** Any information system needs for an infrastructure mechanism that manages data with common services to store, update, remove, retrieve and list general data objects.

The Persistence Manager component works persisting data objects, including the assets content and its meta-information. It abstracts for the consumer of its interfaces where and how the data are stored. It shall allow different kinds of persistence, such as database or file system persistence, as well as dealing with external resources, making the localization of each asset transparent.

- **Artifact Manager:** In the reference architecture, a reusable software unit is represented by a generic element called asset, wide model includes two parts: the asset contents (set of reusable artifacts) and the asset meta-data (asset description). Since the asset insertion operation includes storage of asset contents (artifacts), it is necessary to implement a component that manages such artifacts.

The Artifact Manager represents a repository of asset contents (set of reusable software artifacts).

- **Asset Catalog:** Asset producers need to make their assets available for consumption and then reuse tools should allow asset insertion operation. The insertion operation means storing the asset contents (artifacts) and the asset meta-data which include information about the asset's classification that is useful to organize an asset catalogue.

The AssetCatalog component is responsible for the insertion of assets.

- **Indexer:** Since large sets of assets are unsuitable for direct manipulation in a reasonable time, specialized data structures should be created for representing such assets' data, thus allowing a faster access to their information. In this sense it is necessary to implement a

mechanism in order to generate such structured data. The most common data structure used for this purpose is known as index.

The Indexer component is responsible for analyzing the available assets (its content and meta-information) and generating an index to be used by the Searcher component.

- **Asset Searcher:** Any reuse tool that works with a large number of assets must provide search mechanisms that allow users to find assets that meet their needs. The search implemented by these mechanisms can be a combination of different search types, such as, Free-text, semantic search, Keyword, and Facet-based classification.

The Asset Searcher component is responsible for searching assets stored in the tool. The search service uses *indexes* provided by the *Indexer* component – also located in searcher module. It is possible to configure what search strategy the tool should support. This flexibility is useful, because the tool can be adapted according to the users' necessities.

Training. The subjects who used the proposed process were trained before the study began. The training took 8 hours, divided into 4 lectures with two hours each, during the course. After that, the students spent more 35 hours and 13 minutes studying the component evaluation process and related papers (Figure 8.1).

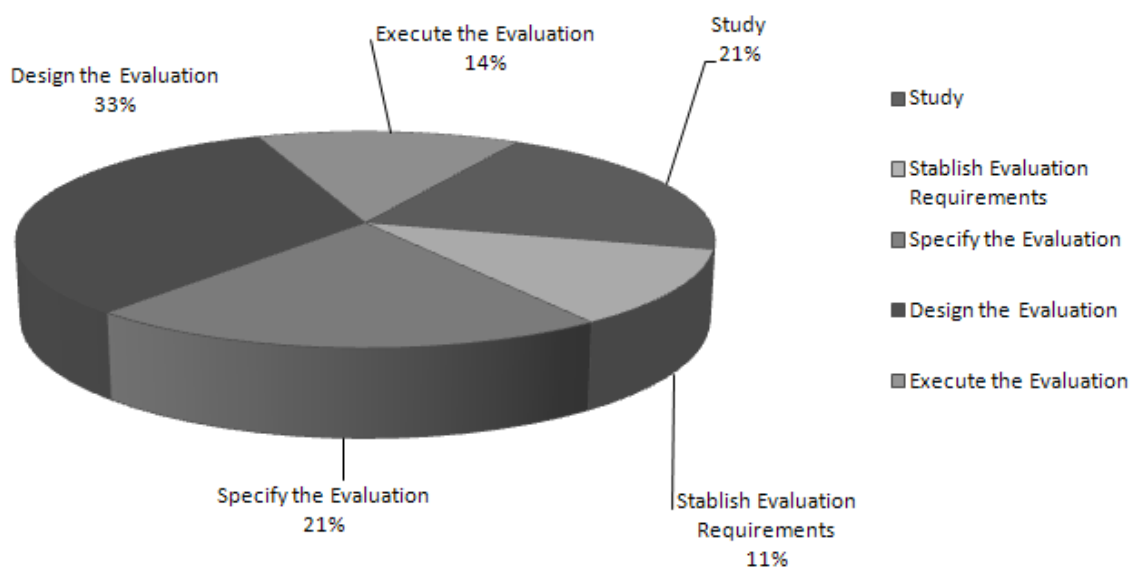


Figure 8.1. Subject's Time Spent in the Experimental Study.

Subjects. The subjects were 3 MS.c. students from Federal University of Pernambuco. All the subjects had industrial experience in software development (more than three years) and quality (one has more than four years and other two subjects have at least one year). Both subjects had participated in industrial projects involving some kind of software quality activity. One subject had training in some issues related to software quality and metrics, such as CMMI, MPS.br, ISO 9001:2000, GQM and Balanced Score Card; and two subjects are specialist of Java programming language and in the Eclipse IDE (i.e. the same language and environment selected to develop the components). Table 8.1 shows a summary of the subjects' profile.

Table 8.1. Subject's Profile in the Experimental Study.

ID	Industrial Projects	Quality Knowledge	Reuse Training
1	- More than 7 large complexity - 1-2 medium complexity - 1-2 small complexity	CMMI, MPS.br and ISO 9001:2000.	Courses: CMMI, MPS.br, and ISO 9001:2000.
2	- 3-7 small complexity	CMMI	-
3	- 3-7 medium complexity - 1-2 small complexity	CMMI and MPS.br	1-2 Conferences

Costs. Since the subjects of the experimental study were students from the Federal University of Pernambuco, and the environment for execution was the university's labs and subject's houses (distributed development), the cost for the study was basically for planning and operation.

8.7 The Analysis and Interpretation

Quantitative Analysis. The quantitative analysis was divided in two analyses: coverage of the component quality attributes and coverage of the evaluation techniques proposed. The analyses were performed using descriptive statistics.

- **Coverage of the component quality attributes.** All components were evaluated based on the following levels: Persistence Manager in SCETM I, Asset Searcher in SCETM II, Asset Catalog in SCETM II, Indexer in SCETM II and Artifact Manager in SCETM I. Those levels were defined through the guidelines for selecting evaluation level presented on chapter 6. The components evaluated at SCETM I used

the characteristics, sub-characteristics and quality attributes presented on Table 8.2.

Table 8.2. Quality Attributes selected for SCETM I.

Characteristics	Sub-Characteristics	Quality Attribute
Functionality	Suitability	Coverage
Reliability	Maturity	Volatility and Failure Removal
Usability	Understandability	Document readability and quality
Usability	Operability	Effort for Operating
Efficiency	Time Behavior	Response Time
Maintainability	Changeability	Customizability
Portability	Deployability	Complexity level
Portability	Reusability	Cohesion and Coupling

The sub-characteristics presented above contain 11 possible quality attributes to be evaluated in a component at SCETM level I (more details about it could be seen at chapter 6 on SCETM). From all of them, the evaluation team selected 10 quality attributes to evaluate the components quality using the SCETM I. Thus, 90.90% of the quality attributes was selected from all of the possible and, in this way, the H_0 was **rejected**.

On the other hand, the components evaluated at SCETM II used the characteristics, sub-characteristics and quality attributes presented on Table 8.3.

Table 8.3. Quality Attributes selected for SCETM II.

Characteristics	Sub-Characteristics	Quality Attribute
Functionality	Accuracy	Correctness
Reliability	Recoverability	Error Handling
Usability	Operability	Provided Interfaces
Usability	Operability	Required Interfaces
Efficiency	Time Behavior	Latency Throughput (“out”)
Efficiency	Time Behavior	Processing Capacity (“in”)
Maintainability	Stability	Modifiability
Portability	Deployability	Complexity level

The sub-characteristics presented above contain 10 possible quality attributes to be evaluated in a component in SCETM level II (more details about it could be seen at chapter 6 on SCETM). From all of them, the evaluation team selected 8 quality attributes to evaluate the components quality using the SCETM II. Thus, 80% of the quality attributes was selected from all of the possible and, in this way, the H_0 was **not rejected**.

This may be because the other two quality attributes (i.e. the quality attributes presented on SCETM level II) not selected (*Failure Removal* and *Backward Compatibility*) were not present in these components, i.e. the components didn't implement any kind of mechanism to remove failures occurring during its execution, and the components do not contain more than one version. For this reason, the evaluation team decided not to consider these quality attributes to evaluate these components quality using SCETM II.

- **Coverage of the evaluation techniques.** After defining the quality attributes, the evaluation team must define which evaluation techniques will be used to measure each quality attribute proposed earlier. Table 8.4 shows the evaluation techniques defined for evaluating components in SCETM I.

Table 8.4. Evaluation Techniques selected to the components evaluated in SCETM I.

Characteristics	Sub-Characteristics	Quality Attribute	Evaluation Techniques
Functionality	Suitability	Coverage	Documentation Analysis
Reliability	Maturity	Volatility and Failure Removal	Suitability and Maturity Analysis
Usability	Understandability	Document readability and quality	Documentation Analysis
Usability	Operability	Effort for Operating	User Mental Model
Efficiency	Time Behavior	Response Time	Evaluation Measurement
Maintainability	Changeability	Customizability	Effort for Operating

Portability	Deployability	Complexity level	Component execution in specific environments analysis
Portability	Reusability	Cohesion and Coupling	Cohesion and Coupling Analysis

The quality attributes presented above contain 11 possible evaluation techniques that should be used to measure the component quality on SCETM level I (more details about it could be seen at chapter 6 on SCETM). From all of them, the evaluation team selected 10 evaluation techniques to evaluate the components quality using SCETM I. Thus, 90.90% of the evaluation techniques was selected and, in this way, the H_0 was **rejected**.

As happen with the H_0 for SCETM I, the H_0 is also rejected once the evaluation techniques selected are the basic techniques for evaluating the quality attribute selected previously (see Table 8.2).

After selecting the quality attributes and the evaluation techniques for SCETM I, the evaluation team should define the GQM, the punctuation level and the tools to be used for each quality attributed in order to execute the evaluation. All data generated during the process are collected in order to be analyzed by the evaluation team.

In this way, the evaluation team measured the *Persistence Manager* and *Artifact Manager* quality using the definitions of the SCETM I, and the quality achieved of those components are presented on Figure 8.2. The results presented could be interpreted as: $0\% \leq x \leq 100\%$; closer to 100% being better.

The results shown in Figure 8.2 may be because the *Data Persistence* component is a basic component implemented for some systems that requires database access.

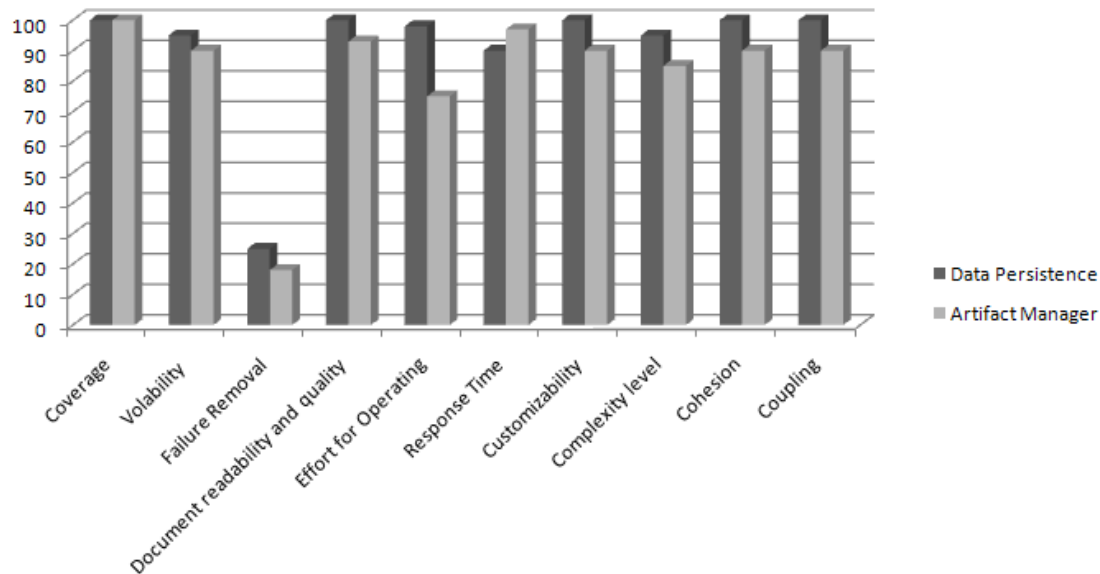


Figure 8.2. Component Quality Measured: Persistence Manager and Artifact Manager.

On the other hand, Table 8.5 shows the evaluation techniques defined the components evaluated using SCETM II.

Table 8.5. Evaluation Techniques selected to the components evaluated in SCMT II.

Characteristics	Sub-Characteristics	Quality Attribute	Evaluation Techniques
Functionality	Accuracy	Correctness	Functional Testing (black box), Unit Test.
Reliability	Recoverability	Error Handling	Programming Language Facilities (Best Practices)
Usability	Operability	Provided Interfaces	Inspection of the interfaces
Usability	Operability	Required Interfaces	Inspection of the interfaces
Efficiency	Time Behavior	Throughput ("out")	Evaluation measurement
Efficiency	Time Behavior	Processing Capacity ("in")	Evaluation measurement
Maintainability	Stability	Modifiability	Inspection of Documents
Portability	Deployability	Complexity level	Deployment analyses

The quality attributes presented above contain 16 possible evaluation techniques that should be used to measure the component quality on SCETM level II (more details about it could be seen at chapter 6 on

SCETM). From all of them, the evaluation team selected 9 evaluation techniques to evaluate the components quality using SCETM II. Thus, 56.25% of the evaluation techniques were selected and, in this way, the H_0 was **not rejected**.

This may be because the quality attributes presented contains a set of evaluation techniques which should be used in different SCETM levels instead of the SCETM II, i.e. some of them are not recommended to use in the SCETM II, for example, Fault tolerance analysis, Reliability growth model, Formal Proof, among others are recommended to use in SCETM III, IV and V levels.

Once selected the quality attributes and the evaluation techniques for SCETM II, the evaluation team defined the GQM, the punctuation level and the tools for each quality attributed in order to execute the evaluation. All data generated during the process are collected in order to be analyzed by the team.

In this way, the evaluation team measured the *Asset Searcher*, *Asset Catalog* and *Indexer* quality using the definitions of the SCETM II, and the quality achieved of those components are presented on Figure 8.3. The results presented could be interpreted as: $0\% \leq x \leq 100\%$; closer to 100% being better.

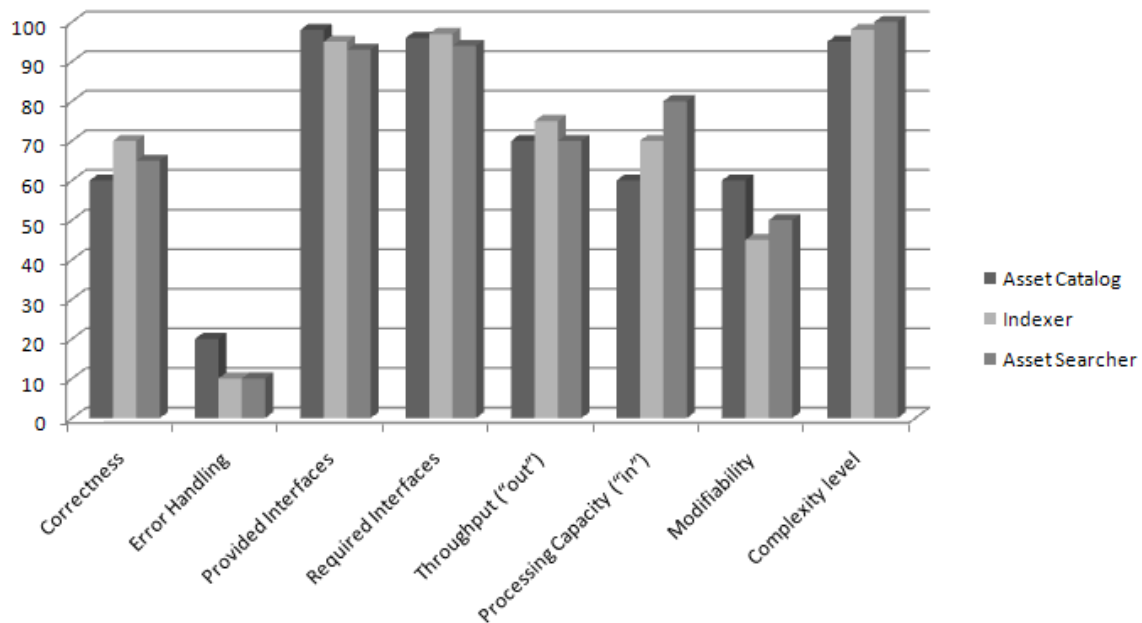


Figure 8.3. Component Quality Measured: Asset Searcher, Asset Catalog and Indexer.

As presented on Table 8.3, both components achieved a similar quality level. This may be due because all subjects that participating of the study (besides the three that participate only from this study presented here) watching all training about the software component quality framework before start the RiDE process usage, i.e. before start the software development. Thus, they may gain knowledge of what is required by the framework in the first levels and looking for implementing the code based on the insights about quality provided by the framework. However, this fact could be considered as best-practice to develop components in order to the developers know what is required for achieving a quality level and implement according to these definitions.

Besides measuring some aspects of the component evaluation process, it is intended to evaluate the difficulties found during the process usage, as presented next.

- **Difficulties in the Component Evaluation Process:** At the end of the study, the subjects answer a questionnaire presented on Appendix C which relates the main difficulties found during the process usage, as show next.
 - **Difficulties to Establish Evaluation Requirement activity.** Analyzing subjects' answers for the difficulties in *establish*

evaluation requirement activity, all subjects related that have some difficulties in the step “*Define the goals/scope of the evaluation*”. In general, the definition of goals and scope of some kinds of software development as Software Product Line (Bayer et al., 1999), (Czarnecki & Eisenecker, 2000), among others, is a challenge for itself.

In this way, in order to decrease this difficulty during the component evaluation process usage it is interesting to store in a knowledge base the past decisions about the goals/scope of the evaluation already executed. Thus, the evaluation team may analyze the past experience to help them during these activities and looking for improving its definitions according to the similarity or relevance of previous goals/scope definition.

- **Difficulties in Specifying the Evaluation activity.** Analyzing the subjects’ answers to the difficulties in *specifying the evaluation activity*, two subjects related that they had some difficulties in which characteristics, sub-characteristics and quality attributes should be selected. This may reflect the software quality education degree of the subjects, which can impact the ability to use the evaluation component process. In other words, the subject that have more experience in software quality does not relate any difficulty to understand the characteristics, sub-characteristics and quality attributes in order to select them during the process.
- **Difficulties in Designing the Evaluation activity.** Analyzing the subjects’ answers to the difficulties in *designing the evaluation activity*, all subjects related that they had some difficulties in defining which evaluation technique should be used to measure the quality attributes defined previously. This may reflects the impossibility to reject the H_0 for SCETM II. On the other words, the high is the SCETM level considered, more is the knowledge of the evaluation team in specific techniques presented on the market in order to evaluate the quality attribute defined.

- **Difficulties in Executing the Evaluation activity.** According to the subjects, they did not have any difficulty during this activity because the whole process was very well documented and it became easy to execute the activities planned.

Among 24 steps from the component evaluation process, the subjects related difficult in only 4 steps, which means 16.6% of difficult during the whole process, and, in this way, the H_0 was **rejected**.

Conclusion. Even with the analysis not being conclusive, the experimental study indicates that the framework is feasible and has a lower complexity level to measure component quality. On the other hand, the aspects related to understanding (i.e. difficulties in activities of the process) need to be reviewed and improved. However, with the results identified in the experiment, the values can be calibrated in a more accurate way. Nevertheless, most of the problems identified by the subjects in terms of difficulties are more related to the provided training than with the process itself. This is further discussed next, in the qualitative analysis.

Qualitative Analysis. After concluding the quantitative analysis for the experiment, the qualitative analysis was performed. This analysis was based on the answers defined for the questionnaire presented in Appendix C.

- **Usefulness of the Process.** All the subjects reported that the process was useful to perform the component quality evaluation. However, all subjects indicated some improvements in both activities of the process which should be carefully considered and reviewed in order to improve the process proposed.
- **Quality of the Material.** Only one subject considered the training insufficient for applying the process. However, all subjects consider very important the background obtained from the lectures related to software component quality, software component evaluation, testing and inspection. All subjects also complained about the lack of examples to clarify the different activities of the process, such as the selection of the quality attributes, the use of the punctuation level and evaluation techniques/tools selection. Some of these aspects were only

related to the background to use the process, but some issues have influenced the difficulty of use, as demonstrated in the quantitative analysis.

8.8 Lessons Learned

After concluding the experimental study, some aspects should be considered in order to repeat the experiment, since they were seen as limitations of the first execution.

Training. Besides the improvements related to the lectures, the subjects highlighted that the training should include a complete and detailed example, covering the whole component evaluation process.

Questionnaires. The questionnaires should be reviewed in order to collect more precise data related to feedback and to the process. Moreover, a possible improvement can be to collect it after the activities during the process usage, avoiding losing useful information by the subjects.

Subjects Skill. The process does not define the skills necessary from each role in the process. Moreover, in this experiment, the roles were defined in an informal way, often allocating the subjects for the roles defined in their jobs. However, this issue should be reviewed in order to be more systematic and to reduce risks.

Subjects knowledge. The subjects that developed the study did not have a considerable experience in software development and quality area. In this way, the results achieved should be better and, consecutively, the framework will be accurately analyzed if the subjects have more experience/knowledge in this area.

8.9 Summary

This chapter presented the definition, planning, operation, analysis and interpretation of the experimental study that evaluate the viability of the component evaluation process. The study analyzed the possibility of subjects using the process to use the CQM (Component Quality Model) and the SCETM (Software Component Technique Model) proposed in this work. The study also

analyzed the difficulties found during the process usage. Besides the results not being conclusive, the experimental study showed that the component quality framework can be used to measure component quality.

The next chapter will present the conclusions of this work, its main contributions, and directions for future works.

9

Conclusions

The growing use of commercial products in large systems makes evaluation and selection of appropriate products an increasingly essential activity. However, many organizations struggle in their attempts to select an appropriate product for use in Component-Based Software Development (CBSD), which is being used in a wide variety of application areas and the correct operations of the components are often critical for business success and, in some cases, human safety. In this way, assessment and evaluation of software components has become a compulsory and crucial part of any CBSD lifecycle. The risk of selecting a product with unknown quality properties is no longer acceptable and, when it happens, it may cause catastrophic results (Jezequel et al., 1997). Thus, software components quality evaluation has become an essential activity in order to bring reliability in (re)using software components.

In this sense, in order to properly enable the evaluation of software components, supplying the real necessities of the software component markets, a Software Component Quality Framework is necessary. Thus, this thesis presented the whole framework and its related-modules: the Component Quality Model (CQM), the Evaluation Techniques Framework represented by the Software Component Techniques Model (SCETM), the Metrics Framework and the Component Evaluation Process. An experimental study was also defined, planned, operated, analyzed and interpreted in order to evaluate the viability of the component evaluation process.

The main goal of this research is to demonstrate that component evaluation is not only possible and practically viable, but also directly applicable in the software industry. In this way, some evaluations have been envisioned in

conjunction to the industry for acquiring trust and maturation to the proposed software component quality framework.

9.1 Research Contributions

The main contributions of this work could be split in three main aspects: **(i)** the realization of a survey related to the state-of-the-art in software component certification research (done during the Master degree and upgraded during the PhD degree); **(ii)** the proposition of a Software Component Quality Framework to evaluate the software component quality; and **(iii)** the accomplishment of an experimental study, in order to evaluate the viability of the proposed framework.

- **A Survey on Software Component Certification.** The main research contributions found in the literature, from the 90's until today, were analyzed in order to understand how the software component certification area has evolved during this timeline. Through this study, it became possible to elaborate a well-defined software component quality framework, consisted of four modules;
- **The Software Component Quality Framework.** The survey showed that software component quality is important to the component market evolution. In order to supply this necessity, a Software Component Quality Framework was defined, which contain a set of modules that complement each other, trying to supply all information required for evaluating the quality of a software component. Each module was defined and discussed with quality experts around the world in order to improve them as much as possible. Moreover, the modules still continue to evolve through its usage in the industry (i.e. RiSE projects); and
- **An Experimental Study.** In order to determine whether the framework meets its proposed goals, an experimental study was performed. This study analyzed the feasibility of the proposed framework, identifying its main drawbacks. The idea is to evaluate how complete are the framework modules provided in order for the evaluation team to execute the whole process and how much effort is needed to use the framework.

9.2 Related Work

Some related works could be found in the literature during this research, according to chapter 4. In this section, a briefly comparison will be presented in relation to those works.

The works presented on chapter 4 were not evaluated neither in academic nor in industrial scenarios, becoming unknown the real efficiency to evaluate software components. The works considering only specific aspects of software component quality (i.e. some researchers works with component quality model, other works with specific kind of software component metrics, other works with component evaluation process, and so on) and don't provide detailed steps that should be carefully followed to accomplish the component evaluation. The works presented a high-level proposal and, in this way, it is very difficult to apply some of them because they don't provide a detailed description about that in order to facilitate its applicability.

Compared to the works described on chapter 4, the software component quality framework proposed in this thesis was developed in the context of a Brazilian software company and was applied, evaluated and tested in a university laboratory in order to evaluate its viability to measure software component quality. Thus, the framework can become more efficient to solve the necessities of the component market (Heineman et al., 2001). Moreover, the framework is composed of four modules that complement each other in the effort to evaluate the component quality level. The steps that should be followed are carefully described in order to facilitate the execution of the process by the evaluation team. The CQM was based on the SQuaRE project (an evolution of the ISO/IEC 9126) and the SCETM is the one of the first proposal model, in literature about evaluation techniques for software components.

9.3 Future Work

Through the results that were obtained (the survey of the state-of-the-art on software component certification area, the proposed software component quality framework and the experimental study), some directions stand up:

- **Cost (Benefits) Model.** An interesting aspect to the customer is the cost/benefit that the component quality assurance could bring to its

business in order to analyze if the costs relative to the component quality assurance are acceptable or not (Keil & Tiwana, 2005). In this way, a Cost (Benefits) Model is very interesting to complement the software component quality framework and should be carefully design to provide as much as possible the real costs and possible benefits to the component customer;

- **Tool Support.** In any engineering discipline it is needed a tool support in order to aid the usage of the proposed processes, methods, techniques, etc. In that way, it is really important the development of tools that support the whole software component quality framework activities once there are a lot of information produced during the evaluation process that could be missed without a tool support;
- **Improve the Framework / Replicate the Experimental Study:** Based on the results of the experimental study, it will be intended to improve the whole framework in order to replicate the experimental study and collect more accurate results of their; and
- **Component Evaluation Center.** The long term plan could be to achieve a degree of maturity that could be used as a component evaluation standard for Software Factories, making it possible to create a Component Evaluation Center (or, perhaps, a standard for component quality). Through the Brazilian projects that the RiSE group is involved, such “dream” may become reality through the maturation of the process and the reliability of the software factories on that process.

9.4 Academic Contributions

The knowledge developed during the work resulted in the following publication:

- **Journals**
 - (Lucrédio et al., 2007) Lucrédio, D.; Brito, K.S.; Alvaro, A.; Garcia, V.C.; Almeida, E.S.; Fortes, R.P.M.; Meira, S.R.L. **Software Reuse: The Brazilian Industry Scenario**, In Journal of Systems and Software (JSS), Elsevier, Vol. 01, No. 06, June, pp. 996-1013, 2008.

- **Books**

- (Almeida et al., 2007b) Almeida, E.S.; Alvaro, A.; Garcia, V.C.; Mascena, J.C.C.P.; Burégio, V.A.; Nascimento, L.M.; Lucrédio, D.; Meira, S.R.L. **C.R.U.I.S.E: Component Reuse in Software Engineering**. C.E.S.A.R e-book, Brazil, 2007.

- **Technical Report**

- (Alvaro et al., 2007e) Alvaro, A.; Land, R.; Crnkovic, I. **Software Component Evaluation: A Theoretical Study on Component Selection and Certification**, In MRTC report ISSN 1404-3041 ISRN MDH-MRTC-217/2007-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, 2007.

- **Conferences**

- (Land et al., 2008) Land, R.; Alvaro, A.; Crnkovic, I. **Towards Efficient Software Component Evaluation: An Examination of Component Selection and Certification**, In the 34st IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Software Process and Product Improvement (SPPI) Track, Parma, Italy, 2008.
- (Alvaro et al., 2007a) Alvaro, A.; Almeida, E.S.; Meira, S. R. L. **Towards a Software Component Certification Framework**, In the 7th IEEE International Conference on Quality Software (QSIC), Portland, Oregon, USA, 2007.
- (Alvaro et al., 2007b) Alvaro, A.; Almeida, E.S.; Meira, S. L. **A Software Component Maturity Model (SCMM)**, In the 33st IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Component-Based Software Engineering (CBSE) Track, Lübeck, Germany, 2007.
- (Alvaro et al., 2007c) Alvaro, A.; Almeida, E. S.; Meira, S.R.L. **Component Quality Assurance: Towards a Software Component Certification Process**. In the IEEE International

Conference on Information Reuse and Integration (IRI), Las Vegas, USA. IEEE Press. 2007.

- (Alvaro et al., 2007d) Alvaro, A.; Almeida, E. S.; Meira, S.R.L. **A Component Quality Assurance Process**, In the Fourth International Workshop on Software Quality Assurance (SOQUA), in conjunction with European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), Dubrovnik, Croatia, 2007.
- (Dias et al., 2007) Dias, J.J.; Cunha, J.A.O.G.; Alvaro, A.; Barros, R.S.M.; Meira, S.R.L. **Web Services Quality Assurance: A XML-based Quality Model**, In the Brazilian Symposium on Software Quality (SBQS), Porto de Galinhas, Brazil, 2007.
- (Oliveira et al., 2007) Oliveira, R.Y.S.; Ferreira, P.G.; Alvaro, A.; Almeida, E.S.; Meira, S. L. **Code Inspection: A Review**, In the 9th International Conference on Enterprise Information Systems (ICEIS), Poster Session, Madeira, Portugal. Lecture Notes in Computer Science (LNCS), Springer-Verlag. 2007.
- (Alvaro et al., 2006a) Alvaro, A.; Almeida, E.S.; Meira, S. R. L. **A Software Component Quality Model: A Preliminary Evaluation**, In the 32st IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Component-Based Software Engineering (CBSE) Track, Cavtat/Dubrovnik, Croatia, 2006.
- (Alvaro et al., 2006b) Alvaro, A.; Almeida, E.S.; Meira, S. L. **Component Quality Model: A Formal Case Study**, In 5th ACM-IEEE International Symposium on Empirical Software Engineering (ISESE), Poster Session, Rio de Janeiro, Brazil, 2006.

The next publications were contributions achieved during the MSc degree in Computer Science, which contributed with the development of this thesis.

- **Dissertation**

- (Alvaro, 2005) Alvaro, A. **Software Component Certification: A Component Quality Model**. MSc. Dissertation, Federal University of Pernambuco, 2005.

- **Conferences**

- (Alvaro et al., 2005a) Alvaro, A.; Almeida, E. S.; Meira, S. R. L. **Software Component Certification: A Survey**. In: The 31st IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Component-Based Software Engineering (CBSE) Track, Porto, Portugal. IEEE Press. 2005.
- (Alvaro et al., 2005b) Alvaro, A.; Almeida, E. S.; Meira, S. R. L. **Towards a Software Component Quality Model**. In: The 31st IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Work in Progress Session, Porto, Portugal, 2005.
- (Alvaro et al., 2005c) Alvaro, A.; Almeida, E. S.; Meira, S. R. L. **Quality Attributes for a Component Quality Model**. In: The 10th International Workshop on Component-Oriented Programming (WCOP) in Conjunction with the 19th European Conference on Object Oriented Programming (ECOOP), Glasgow, Scotland. 2005.
- (Alvaro et al., 2005d) Alvaro, A.; Almeida, E. S.; Meira, S. R. L. **Component Certification: A Component Quality Model**. In: The III Workshop de Teses e Dissertações em Qualidade de Software (WTDQS) in Conjunction with the 4th Simpósio Brasileiro de Qualidade de Software (SBQS), Porto Alegre, Brazil. 2005.
- (Almeida et al., 2005) Almeida, E. S.; Alvaro, A.; Garcia, V.C.; Lucrédio, D.; Meira, S. R. L. **A Survey on Software Reuse Processes**. In: The IEEE International Conference on Information Reuse and Integration (IRI), Las Vegas, Nevada, USA. IEEE Press. 2005.

- (Almeida et al., 2004a) Almeida, E. S.; Alvaro, A.; Garcia, V.C.; Lucrédio, D.; Meira, S. R. L. **RiSE Project: Towards a Robust Framework for Software Reuse**. In: The IEEE International Conference on Information Reuse and Integration (IRI), Las Vegas, Nevada, USA. IEEE Press. 2004.

9.5 Other Publications

Besides the results listed above, there were other publications during the period of this work, not directly related to the subject of this dissertation but an important experience during the research.

- **Journals**

- (Almeida et al., 2007c) Almeida, E.S.; Alvaro, A.; Garcia, V. C.; Nascimento, L. M.; Lucrédio, D.; Meira, S. R. L. **A Systematic Approach to Design Domain-Specific Software Architectures**, Journal of Software, Academy Publisher, August, Vol.02, No.02, 2007.

- **Books**

- (Ramos et al., 2007a) Ramos, R.A.; Silva, J.; Alvaro, A.; Afonso, R.A. **PHP para Profissionais (in portuguese)**, Editora Digerati, ISBN: 978-85-60480-64-7, 2007.
- (Ramos et al., 2007b) Ramos, R.A.; Silva, J.; Alvaro, A.; **Curso Essencial de VBA (in portuguese)**, Editora Digerati, ISBN: 978-85-60480-67-8, 2007.

- **Conferences**

- (Almeida et al., 2008a) Almeida, E.S.; Alvaro, A.; Garcia, V.C.; Lucrédio, D.; Fortes, R.P.M.; Meira, S.R.L. **A Systematic Process for Domain Engineering**, The 20th International Conference on Software Engineering and Knowledge Engineering (SEKE), USA, 2008.
- (Almeida et al., 2008b) Almeida, E.S.; Santos, E.C.R.; Alvaro, A.; Garcia, V. C.; Lucrédio, D.; Fortes, R.P.M.; Meira, S. R. L. **Domain Implementation in Software Product Lines Using OSGi**, In

the 7th IEEE International Conference on COTS-Based Software Systems (ICCBSS), 2008.

- (Almeida et al., 2007d) Almeida, E.S.; Alvaro, A.; Garcia, V.C.; Lucrédio, D.; Fortes, R.P.M.; Meira, S. L. **An Experimental Study in Domain Engineering**, In the 33st IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Component-Based Software Engineering (CBSE) Track, Lübeck, Germany, 2007.
- (Almeida et al., 2007e) Almeida, E.S.; Alvaro, A.; Lucrédio, D.; Garcia, V.C.; Nascimento, L.M.; Meira, S. L. **Designing Domain-Specific Software Architecture (DSSA): Towards a New Approach**, In 6th Working IEEE/IFIP Conference on Software Architecture, Mumbai, India, 2007.
- (Brito et al., 2006) Brito, K.S.; Alvaro, A.; Lucrédio, D.; Almeida, E.S.; Meira, S. L. **Software Reuse: A Brief Overview of the Brazilian Industry's Case**, In 5th ACM-IEEE International Symposium on Empirical Software Engineering (ISESE), Short Paper, Rio de Janeiro, Brazil, 2006.
- (Cavalcanti et al., 2006) Cavalcanti, A.P.C.; Alvaro, A.; Almeida, E.S.; Meira, S.R.L. **Reuse Process Adaptation Strategies**. In the 32st IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Component-Based Software Engineering (CBSE) Track, Cavtat/Dubrovnik, Croatia, 2006.

9.6 Summary

The advent of software components has dramatically changed the way that the software industry develops its software systems, increasing the productivity and the quality of the software produced. Among these powerful improvements, software component technology has become an economic necessity because it shortens the implementation timeline and lessens the unpredictability associated with developing custom application. However, the functionality and quality of the selected components are, actually, the main concerns related to

the software engineers and managers of some software industries around the world (Keil et al., 2005).

Motivated by these reasons, this thesis proposes a software component quality framework in order to establish the relevant information important to evaluate the component quality, and presents an experimental study, which evaluate the viability of the application of the software component quality framework

The target is, in conjunction with the industry, to investigate the component quality evaluation area in order to evolve the following modules proposed on the framework: **(i)** a Component Quality Model, **(ii)** a Evaluation Techniques Framework, **(iii)** a Metrics Framework, and **(iv)** a Component Evaluation Process. As previously cited, this project is part of RiSE project, whose main concerns are: developing a robust framework for software reuse (Almeida et al., 2004a), in order to establish a standard to the component development; and defining and developing a repository system and a component evaluation process.

Based on this software component quality framework, the long term plan is to create a Component Evaluation Center in order to provide a place for assuring the quality of the software components provided by the markets and the software industry.



References

(Almeida et al., 2008a) Almeida, E.S.; Alvaro, A.; Garcia, V.C.; Lucrédio, D.; Fortes, R.P.M.; Meira, S.R.L. **A Systematic Process for Domain Engineering**, The 20th International Conference on Software Engineering and Knowledge Engineering (SEKE), USA, 2008.

(Almeida et al., 2008b) Almeida, E.S.; Santos, E.C.R.; Alvaro, A.; Garcia, V. C.; Lucrédio, D.; Fortes, R.P.M.; Meira, S. R. L. **Domain Implementation in Software Product Lines Using OSGi**, In: *The 7th IEEE International Conference on COTS-Based Software Systems (ICCBSS)*, 2008.

(Almeida, 2007) Almeida, E.S. **The RiSE Process for Domain Engineering (RIDE)**, *PhD. Thesis*, Federal University of Pernambuco, 2007.

(Almeida et al., 2007a) Almeida, E. S.; Alvaro, A.; Meira, S. R. L. **Key Developments in the Field of Software Reuse**, In: *Submitted to the ACM Computing Surveys*, 2007.

(Almeida et al., 2007b) Almeida, E.S.; Alvaro, A.; Garcia, V.C.; Mascena, J.C.C.P.; Burégio, V.A.; Nascimento, L.M.; Lucrédio, D.; Meira, S.R.L. **C.R.U.I.S.E: Component Reuse in Software Engineering**. *C.E.S.A.R e-book*, Brazil, 2007.

(Almeida et al., 2007c) Almeida, E.S.; Alvaro, A.; Garcia, V. C.; Nascimento, L. M.; Lucrédio, D.; Meira, S. R. L. **A Systematic Approach to Design Domain-Specific Software Architectures**, In: *Journal of Software*, Academy Publisher, August, Vol.02, No.02, 2007.

- (Almeida et al., 2007d) Almeida, E.S.; Alvaro, A.; Garcia, V.C.; Lucrédio, D.; Fortes, R.P.M.; Meira, S. L. **An Experimental Study in Domain Engineering**, In: *The 33rd IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Component-Based Software Engineering (CBSE) Track*, Lübeck, Germany, 2007.
- (Almeida et al., 2007e) Almeida, E.S.; Alvaro, A.; Lucrédio, D.; Garcia, V.C.; Nascimento, L.M.; Meira, S. L. **Designing Domain-Specific Software Architecture (DSSA): Towards a New Approach**, In: *The 6th Working IEEE/IFIP Conference on Software Architecture*, Mumbai, India, 2007.
- (Almeida et al., 2005) Almeida, E. S.; Alvaro, A.; Garcia, V.C.; Lucrédio, D.; Meira, S. R. L. **A Survey on Software Reuse Processes**. In: *The IEEE International Conference on Information Reuse and Integration (IRI)*, Las Vegas, Nevada, USA. IEEE Press. 2005.
- (Almeida et al., 2004) Almeida, E. S.; Alvaro, A.; Lucrédio, D.; Garcia, V. C.; Meira, S. R. L. **RiSE Project: Towards a Robust Framework for Software Reuse**, In: *The IEEE International Conference on Information Reuse and Integration (IRI)*, Las Vegas, USA, pp. 48-53, 2004.
- (Alvaro et al., 2007a) Alvaro, A.; Almeida, E.S.; Meira, S. R. L. **Towards a Software Component Certification Framework**, In: *The 7th IEEE International Conference on Quality Software (QSIC)*, Portland, Oregon, USA, 2007.
- (Alvaro et al., 2007b) Alvaro, A.; Almeida, E.S.; Meira, S. R. L. **A Software Component Maturity Model (SCMM)**, In: *The 33rd IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Component-Based Software Engineering (CBSE) Track*, Lübeck, Germany, 2007.
- (Alvaro et al., 2007c) Alvaro, A.; Almeida, E. S.; Meira, S. R. L. **Component Quality Assurance: Towards a Software Component Certification Process**. In: *The IEEE International Conference on*

- Information Reuse and Integration (IRI)*, Las Vegas, USA. IEEE Press. 2007.
- (Alvaro et al., 2007d) Alvaro, A.; Almeida, E. S.; Meira, S.R.L. **A Component Quality Assurance Process**. In: *The Fourth International Workshop on Software Quality Assurance (SOQUA)*, in conjunction with European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), Dubrovnik, Croatia, 2007.
- (Alvaro et al., 2007e) Alvaro, A.; Land, R.; Crnkovic, I. **Software Component Evaluation: A Theoretical Study on Component Selection and Certification**, In: *MRTC report*, ISSN 1404-3041 ISRN MDH-MRTC-217/2007-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, 2007.
- (Alvaro et al., 2006a) Alvaro, A.; Almeida, E.S.; Meira, S. R. L. **A Software Component Quality Model: A Preliminary Evaluation**, In: *The 32st IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Component-Based Software Engineering (CBSE) Track*, Cavtat/Dubrovnik, Croatia, 2006.
- (Alvaro et al., 2006b) Alvaro, A.; Almeida, E.S.; Meira, S. L. **Component Quality Model: A Formal Case Study**, In: *5th ACM-IEEE International Symposium on Empirical Software Engineering (ISESE)*, Poster Session, Rio de Janeiro, Brazil, 2006.
- (Alvaro, 2005) Alvaro, A. **Software Component Certification: A Component Quality Model**, *MSc. Dissertation*, Federal University of Pernambuco, 2005.
- (Alvaro et al., 2005a) Alvaro, A.; Almeida, E. S.; Meira, S. R. L. **A Software Component Certification: A Survey**, In: *The 31st IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Component-Based Software Engineering (CBSE) Track*, Porto, Portugal, 2005.
- (Alvaro et al., 2005b) Alvaro, A.; Almeida, E. S.; Meira, S. R. L. **Towards a Component Quality Model**, In: *The 31st IEEE EUROMICRO*

- Conference on Software Engineering and Advanced Applications (SEAA), Work in Progress Session*, Porto, Portugal, 2005.
- (Alvaro et al., 2005c) Alvaro, A.; Almeida, E. S.; Meira, S. R. L. **Quality Attributes for a Component Quality Model**, In: *The 10th International Workshop on Component Oriented Programming (WCOP) in conjunction with the 19th ACM European Conference on Object Oriented Programming (ECCOP)*, Glasgow, Scotland, 2005.
- (Alvaro et al., 2005d) Alvaro, A.; Meira, S. R. L. **Component Certification: A Component Quality Model**. In: *The III Workshop de Teses e Dissertações em Qualidade de Software*, Porto Alegre, Brazil, 2005.
- (Andreou & Tziakouris, 2007) Andreou, A. S.; Tziakouris, M. **A quality framework for developing and evaluating original software components**. In: *Information & Software Technology*. Vol. 49, No. 02, pp. 122-141, 2007.
- (Arango, 1994) Arango, G. **A Brief Introduction to Domain Analysis**, In: *ACM symposium on Applied Computing*, USA, pp. 42-46, 1994.
- (Barachisio et al., 2008) Barachisio, L.; Nascimento, L.; Almeida, E.S.; Meira, S.R.L. **A Case Study in Software Product Lines: An Educational Experience**, In: *21st IEEE-CS Conference on Software Engineering Education and Training*, EUA, 2008.
- (Barros et al., 2002) Barros, M. O.; Werner, C. M. L.; Travassos, G. H. **An Experimental Study about Modeling Use and Simulation in support to the Software Project Management (in portuguese)**, In: *The 16th Brazilian Symposium in Software Engineering*, Rio de Janeiro, Brazil, 2002.
- (Basili et al., 1996) Basili, V. R.; Briand, L. C.; Melo, W. L. **How reuse influences productivity in object-oriented systems**. In: *Communications of the ACM*, Vol. 39, No. 10, pp. 104-116, 1996.
- (Basili et al., 1996b) V. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Sorumgaard, and M. Zelkowitz, **Packaging Researcher Experience to Assist Replication of Experiments**, Sydney, Australia ISERN Meeting, 1996.

- (Basili et al., 1994) Basili, V.R.; Caldiera, G.; Rombach, H.D. **The Goal Question Metric Approach**, In: *Encyclopedia of Software Engineering*, Vol. II, September, pp. 528-532, 1994.
- (Basili, 1992) Basili, V.R. **Software Modeling and Measurement: The Goal Question Metric Paradigm**, In: *Computer Science Technical Report Series*, CS-TR-2956 (UMIACS-TR-92-96), University of Maryland, College Park, MD, September 1992.
- (Basili et al., 1991) Basili, V. R.; Rombach, H. D. **Support for comprehensive reuse**, In: *IEEE Software Engineering Journal*, Vol. 06, No. 05, pp. 303-316, 1991.
- (Basili et al., 1986) Basili, V. R.; Selby, R.; Hutchens, D. **Experimentation in Software Engineering**, In: *IEEE Transactions on Software Engineering*, Vol. 12, No. 07, pp. 733-743, 1986.
- (Basili & Rombach, 1988) Basili, V.R.; Rombach, H.D. **The TAME Project: Towards Improvement-Oriented Software Environments**, In: *IEEE Transactions on Software Engineering*, Vol. 14, No.6, pp.758-773, 1988.
- (Basili & Selby, 1984) Basili, V.R.; Selby, R.W. **Data Collection and Analysis in Software Research and Management**, In: *Proceedings of the American Statistical Association and Biomeasure Society*, Joint Statistical Meetings, Philadelphia, 1984.
- (Basili & Weiss, 1984) Basili, V.R.; Weiss, D.M. **A Methodology for Collecting Valid Software Engineering Data**, In: *IEEE Transactions on Software Engineering*, Vol. 10, No. 06, pp. 728-738, 1984.
- (Bass et al., 2000) Bass, L.; Buhman, C.; Dorda, S.; Long, F.; Robert, J.; Seacord, R.; Wallnau, K. C. **Market Assessment of Component-Based Software Engineering**, In: *Software Engineering Institute (SEI), Technical Report*, Vol. I, May, 2000.
- (Bay and Pauls, 2004) Bay, T.G.; Pauls, K. **Reuse Frequency as Metric for Component Assessment**. In: *Technical Report 464*, ETH Zürich, Chair of Software Engineering, 2004.

- (Bayer et al., 1999) J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, J. DeBaud, **PuLSE: A Methodology to Develop Software Product Lines**, In: *Symposium on Software Reusability (SSR)*, Los Angeles, USA, May, 1999, pp. 122-131.
- (Bertoa & Vallecillo, 2004) Bertoa, M. F.; Vallecillo, A. **Usability Metrics for Software Components**, In: *Proceedings of the 8th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE)*, Oslo, Norway, June 2004.
- (Bertoa et al., 2006) Bertoa, M.F.; Troya, J.M.; Vallecillo, A. **Measuring the Usability of Software Components**. In: *Journal of Systems and Software*, Vol. 79, No. 03, pp. 427-439, 2006.
- (Bertoa et al., 2003) Bertoa, M. F.; Troya, J. M.; Vallecillo, A. **A Survey on the Quality Information Provided by Software Component Vendors**, In: *The Proceedings of the 7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE)*, Germany, July, 2003.
- (Bertoa et al., 2002) Bertoa, M.; Vallecillo, A. **Quality Attributes for COTS Components**, In: *The 6th IEEE International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE)*, Spain, Vol. 01, No. 02, pp. 128-144, 2002.
- (Bertolino & Mirandola, 2003) Bertolino, A.; Mirandola, R. **Towards Component-Based Software Performance Engineering**, In: *Proceedings of 6th ICSE Workshop on Component-Based Software Engineering*, USA, 2003.
- (Beugnard et al., 1999) Beugnard, A.; Jezequel, J.; Plouzeau, N.; Watkins, D. **Making component contract aware**, In: *IEEE Computer*, Vol. 32, No. 07, pp. 38-45, 1999.
- (Beus-Dukic et al., 2003) Beus-Dukic, L.; Boegh, J. **COTS Software Quality Evaluation**, In: *The 2nd International Conference on COTS-Based Software System (ICCBSS)*, Lecture Notes in Computer Science (LNCS), Springer-Verlag, Canada, 2003.

- (Beydeda & Gruhn, 2003) Beydeda, S.; Gruhn, V. **State of the art in testing components**, In: *The 3th IEEE International Conference on Quality Software (ICQS)*, USA, 2003.
- (Boegh et al., 1993) Boegh, J.; Hausen, H-L.; Welzel, D. **A Practioners Guide to Evaluation of Software**, In: *The IEEE Software Engineering Standards Symposium*, pp. 282-288, 1993.
- (Boehm et al., 1978) Boehm, B.; Brown, J.R.; Lipow, H.; MacLeod, G. J.; Merrit, M. J. **Characteristics of Software Quality**, Elsevier North Holland, 1978.
- (Boehm et al., 1976) Boehm, W.; Brown, J.R.; Lipow, M. **Quantitative Evaluation of Software Quality**, In: *The Proceedings of the Second International Conference on Software Engineering*, pp.592-605, 1976.
- (Boer et al., 2002) Boer, F.S. de; Bonsangue, M.; Graf, S.; de Roever, W.P. **Formal Methods for Components and Objects**, In *the First International Symposium FMCO*, Lecture Notes in Computer Science, Leiden, The Netherlands, Vol. 2852, pp. 509, 2002.
- (Brereton et al., 2000) Brereton, P.; Budgen, D. **Component-Based Systems: A Classification of Issues**, In: *IEEE Computer*, Vol. 33, No. 11, pp. 54-62, 2000.
- (Brownsword et al., 2000) Brownsword, L.; Oberndorf, T.; Sledge, C. A.: **Developing New Processes for COTS-Based Systems**. In: *IEEE Software*, July/August, pp. 48-55, 2000.
- (Caldiera & Basili, 1991) Caldiera, G.; Basili, V. **Identifying and Qualifying Reusable Software Components**, In: *IEEE Computer*, Vol. 24, No. 02, pp. 61–71, 1991.
- (Cavalcanti et al., 2006) Cavalcanti, A.P.C.; Alvaro, A.; Almeida, E.S.; Meira, S.R.L. **Reuse Process Adaptation Strategies**. In: *The 32st IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Component-Based Software Engineering (CBSE) Track*, Cavtat/Dubrovnik, Croatia, 2006.

- (Cechich et al., 2003) Cechich, A.; Vallecillo, A.; Piattini, M. **Assessing component based systems**, In: *Component Based Software Quality*, Lecture Notes in Computer Science (LNCS), pp. 1–20, 2003.
- (Chen et al., 2005) Chen, S.; Liu, Y.; Gorton, I.; Liu, A. **Performance Predication of Component-based Applications**. In: *Journal of Systems and Software*, Vol. 01, No. 07, pp. 35-46, 2005.
- (Cho et al., 2001) Cho, E. S.; Kim, M. S.; Kim, S. D. **Component Metrics to Measure Component Quality**, In: *The 8th IEEE Asia-Pacific Software Engineering Conference (APSEC)*, pp. 419-426, 2001.
- (Choi et al., 2008) Choi, Y.; Lee, S.; Song, H.; Park, J.; Kim, S. **Practical S/W Component Quality Evaluation Model**, In: *The 10th IEEE International Conference on Advanced Communication Technology (ICACT)*, Korea, 2008.
- (Clements et al., 2001) Clements, P.; Northrop, L. **Software Product Line: Practices and Patterns**, In: *SEI Series in Software Engineering*, Addison Wesley, USA, 2001.
- (CMMI, 2006) CMMI for Development. **CMMI for Development, Version 1.2**, In: Technical Report *CMU/SEI-2006-TR-008*, Carnegie Mellon University/Software Engineering Institute (CMU/SEI), 2006.
- (Comella-Dorda et al., 2002) Comella-Dorda, S.; Dean, J.; Morris, E.; Oberndorf, P. **A Process for COTS Software Product Evaluation**, In: *The 1st International Conference on COTS-Based Software System (ICCBSS)*, Lecture Notes in Computer Science (LNCS), Springer-Verlag, USA, 2002.
- (Comella-Dorda et al., 2003) Comella-Dorda, S.; Dean, J.; Lewis, G.; Morris, E.; Oberndorf, P.; Harper, E. **A Process for COTS Software Product Evaluation**, In: *Technical Report*, CMU/SEI-2003-TR-017, 2003.
- (Councill, 1999) Councill, W. T. **Third-Party Testing and the Quality of Software Components**, In: *IEEE Computer*, Vol. 16, No. 04, pp. 55-57, 1999.

- (Councill et al., 2000) Councill, W.T.; Flynt, J. S.; Mehta, A.; Speed, J. R.; Shaw, M. **Component-Based Software Engineering and the Issue of Trust**, In: *The 22th IEEE International Conference on Software Engineering (ICSE)*, Ireland, pp. 661-664, 2000.
- (Councill, 2001) Councill, B. **Third-Party Certification and Its Required Elements**, In: *The 4th Workshop on Component-Based Software Engineering (CBSE)*, Lecture Notes in Computer Science (LNCS), Springer-Verlag, Canada, May, 2001.
- (Crnkovic et al., 2002) Crnkovic, I.; Schmidt H.; Stafford J.; Wallnau K. C. **Proc. of the 5th Workshop on Component-Based Software Engineering(CBSE): Benchmarks for Predictable Assembly**, In: *The Software Engineering Notes*, Vol. 27, No. 05, 2002.
- (Crnkovic et al., 2001) Crnkovic, I.; Schmidt H.; Stafford J.; Wallnau K. C. **Proc. of the 4th Workshop on Component-Based Software Engineering(CBSE): Component Certification and System Prediction**, In: *The Software Engineering Notes*, Vol 26, No. 06, 2001.
- (Crnkovic, 2001) Crnkovic, I. **Component-based software engineering - new challenges in software development**, In: *Software Focus*, Vol. 02, No. 04, pp. 27-33, 2001.
- (Czarnecki & Eisenecker, 2000) K. Czarnecki, U. W. Eisenecker, **Generative Programming: Methods, Tools, and Applications**, Addison-Wesley, 2000, pp. 832.
- (DeMichiel, 2002) DeMichiel, L. G. **Enterprise JavaBeans (EJB) Specification**, Version 2.1, Sun Microsystems, 2002.
- (Drouin, 1995) Drouin, J-N. **The SPICE Project: An Overview**. In: *The Software Process Newsletter, IEEE TCSE*, No. 02, pp. 08-09, 1995.
- (D'Souza et al., 1999) D'Souza, D. F.; Wills, A. C. **Objects, Components, and Frameworks with UML, The Catalysis Approach**. Addison-Wesley, USA, 1999.

- (Endres, 1993) Endres, A. **Lessons Learned in an Industrial Software Lab**, In: *IEEE Software*, Vol. 10, No. 05, pp. 58-61, 1993.
- (Fagan, 1976) Fagan, M. **Design and Code Inspections to Reduce Errors in Program Development**. In: *IBM Systems Journal*, Vol. 15, No. 03, pp. 182-211, 1976.
- (Farroq & Dominick, 1988) Farooq, M.U.; & Dominick, W.D. **A survey of formal tools and models for developing user interfaces**, In the *International Journal of Man-Machine Studies*, Vol. 29, No. 05, pp. 479-496, 1988.
- (Frakes & Terry, 1996) Frakes, W.; Terry, C. **Software Reuse: Metrics and Models**, In: *ACM Computing Survey*, Vol. 28, No. 02, pp. 415-435, 1996.
- (Frakes & Fox, 1995) Frakes, W. B.; Fox, S. **Sixteen Questions about Software Reuse**, In: *Communications of the ACM*, Vol. 38, No. 06, pp. 75-87, 1995.
- (Frakes & Isoda, 1994) Frakes, W. B.; Isoda, S. **Success Factors of Systematic Reuse**, In: *IEEE Software*, Vol. 11, No. 05, pp. 15-19, 1994.
- (Freedman, 1991) Freedman, R.S. **Testability of Software Components**, In: *IEEE Transactions on Software Engineering*, Vol. 17, No. 06, June 1991.
- (Fusario et al., 1997) Fusario, P., Lanubile, F., and Visaggio, G. **A Replicated Experiment to Assess Requirements Inspection Techniques**, In: *Empirical Software Engineering: An International Journal*, Vol. 02, No. 01, pp. 39-57, 1997.
- (Gamma et al., 1995) Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison-Wesley, 1995.
- (Gao et al., 2003) Gao, J.Z.; Jacob, H.S.J.; Wu, Y. **Testing and Quality Assurance for Component Based Software**, Artech House, 2003.

- (Georgiadou, 2003) Georgiadou, E. **GEQUAMO-A Generic, Multilayered, Customisable, Software Quality Model**. In: *Software Quality Journal*, Vol. 11, No. 04, pp. 313-323, 2003.
- (Goulao et al., 2002a) Goulao, M.; Brito e Abreu, F. **The Quest for Software Components Quality**, In: *The 26th IEEE Annual International Computer Software and Applications Conference (COMPSAC)*, England, pp. 313-318, 2002.
- (Goulão et al., 2002b) Goulão, M.; Abreu, F. B. **Towards a Component Quality Model**, In: *The 28th IEEE EUROMICRO Conference, Work in Progress Session*, Dortmund, Germany, 2002.
- (Griss et al., 1995) Griss, M. L.; Wosser, M.; Pfleeger, S. L. **Making Software Reuse Work at Hewlett-Packard**, In: *IEEE Software*, Vol. 12, No. 01, pp. 105-107, 1995.
- (Griss, 1994) Griss, M. L. **Software reuse experience at Hewlett-Packard**, In: *The 16th IEEE International Conference on Software Engineering (ICSE)*, Italy, pp. 270, 1994.
- (Gui & Scott, 2007) Gui, G.; Scott, P.D. **Ranking reusability of software components using coupling metrics**. In: *Journal of Systems and Software*, Vol. 80, No. 09, pp. 1450-1459, 2007.
- (Hall, 1990) Hall, A., **Seven Myths of Formal Methods**, In: *IEEE Software*, pp. 11-20, 1990.
- (Hamlet et al., 2001) Hamlet, D.; Mason, D.; Woit. D. **Theory of Software Component Reliability**, In: *23rd International Conference on Software Engineering (ICSE)*, 2001.
- (Hatton, 2007) Hatton, L. **The Chimera of Software Quality**, In *IEEE Computer*, August 2007.
- (Heineman et al., 2001) Heineman, G. T.; Councill, W. T. **Component-Based Software Engineering: Putting the Pieces Together**, Addison-Wesley, USA, 2001.

- (Hissam et al., 2003) Hissam, S. A.; Moreno, G. A.; Stafford, J.; Wallnau, K. C. **Enabling Predictable Assembly**, In: *Journal of Systems and Software*, Vol. 65, No. 03, pp. 185-198, 2003.
- (Hyatt et al., 1996) Hyatt, L.; Rosenberg, L.; **A Software Quality Model and Metrics for Risk Assessment**, In: *NASA Software Technology Assurance Center (SATC)*, 1996.
- (ISO/IEC 25000, 2005) ISO/IEC 25000, **Software product quality requirements and evaluation (SQuaRE)**, Guide to SQuaRE, International Standard Organization, July, 2005.
- (ISO/IEC 25010) ISO/IEC 25020, **Software product Quality Requirements and Evaluation (SQuaRE) — Quality model (in elaboration)**.
- (ISO/IEC 15939, 2007) ISO/IEC 15939, **Software Engineering - Software Measurement Process**, 2007.
- (ISO/IEC 25020, 2007) ISO/IEC 25020, **Software product Quality Requirements and Evaluation (SQuaRE) — Quality measurement — Measurement reference model and guide**, 2007.
- (ISO/IEC 25030, 2007) ISO/IEC 25030, **Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Quality requirements**, 2007.
- (ISO/IEC 25040) ISO/IEC 25040, **Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Evaluation reference model and guide (in elaboration)**.
- (ISO/IEC 25051, 2006) ISO 25051, **Requirements for quality of Commercial Off-The-Shelf (COTS) software product and instructions for testing — Product Quality Requirements and Evaluation (SQuaRE)**, International Standard ISO/IEC 25051, International Standard Organization (ISO), 2006.

- (ISO/IEC 9126, 2001) ISO 9126, **Information Technology – Product Quality – Part1: Quality Model**, International Standard ISO/IEC 9126, International Standard Organization (ISO), 2001.
- (ISO/IEC 9126-2, 2003) ISO 9126-2, **Information Technology – Software product evaluation -- Part 2: External Metrics**, International Standard ISO/IEC 9126-2, International Standard Organization (ISO), 2003.
- (ISO/IEC 9126-3, 2003) ISO 9126-3, **Information Technology – Software product evaluation -- Part 3: Internal Metrics**, International Standard ISO/IEC 9126-3, International Standard Organization (ISO), 2003.
- (ISO/IEC 9126-4, 2003) ISO 9126-4, **Information Technology – Software product evaluation -- Part 4: Quality in Use Metrics**, International Standard ISO/IEC 9126-4, International Standard Organization (ISO), 2003.
- (ISO/IEC 14598, 1998) ISO 14598, **Information Technology – Software product evaluation -- Part 1: General Guide**, International Standard ISO/IEC 14598, International Standard Organization (ISO), 1998.
- (ISO/IEC 9000, 2005) ISO/IEC 9000, **Quality Management Systems – Fundamentals and vocabulary**, International Standard ISO/IEC 9000, International Standard Organization (ISO), 2005.
- (ISO/IEC 15504-7, 2003) ISO/IEC 15504-7, **Information technology – Process ssessment – Part 7 : Assessment of organizational maturity**, International Standard ISO/IEC 15504-7, International Standard Organization (ISO), 2008.
- (Jacobson et al., 1997) Jacobson, I.; Griss, M.; Jonsson, P. **Software Reuse: Architecture, Process and Organization for Business Success**, Addison-Wesley, Longman, 1997.
- (Jezequel et al., 1997) Jezequel, J. M.; Meyer, B. **Design by Contract: The Lessons of Ariane**, In: *IEEE Computer*, Vol. 30, No. 02, pp. 129-130, 1997.

- (Joos, 1994) Joos, R. **Software Reuse at Motorola**, In: *IEEE Software*, Vol. 11, No. 05, pp. 42-47, 1994.
- (Kallio et al., 2001) Kallio, P.; Niemelä, E. **Documented quality of COTS and OCM components**, In: *The 4th Workshop on Component-Based Software Engineering (CBSE)*, Lecture Notes in Computer Science (LNCS) Springer-Verlag, USA, 2001.
- (Kazman et al., 2000) Kazman, R.; Klein, M.; Clements, P. **ATAM: Method for Architecture Evaluation**, In: *The Technical Report CMU/SEI-2000-TR-004*, 2000.
- (Keil & Tiwana, 2005) Keil, M; Tiwana, A. **Beyond Cost: The Drivers of COTS Application Value**. In: *IEEE Software*, Vol. 22, No. 03, pp. 64-69, 2005.
- (Kogure and Akao, 1983) Kogure, M.; Akao, Y. **Quality Function Deployment and CWQC in Japan**, In: *Quality Progress*, pp.25-29, 1983.
- (Kotula, 1998) Kotula, J. **Using Patterns To Create Component Documentation**, In: *IEEE Software*, Vol. 15, No. 02, March/April, pp. 84-92, 1998.
- (Krueger, 1992) Krueger, C. W. **Software Reuse**, In: *ACM Computing Surveys*, Vol. 24, No. 02, pp. 131-183, 1992.
- (Land et al., 2008) Land, R.; Alvaro, A.; Crnkovic, I. **Towards Efficient Software Component Evaluation: An Examination of Component Selection and Certification**, In: *The 34st IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Software Process and Product Improvement (SPPI) Track*, Parma, Italy, 2008.
- (Lethbridge et al., 2003) Lethbridge, T.; Singer, J.; Forward, A. **How Software Engineers use Documentation: The State of the Practice**, In: *IEEE Software*, Vol. 20, No. 06, pp. 35-39, 2003.
- (Lim, 1994) Lim, W. C. **Effects of Reuse on Quality, Productivity, and Economics**, In: *IEEE Software*, Vol. 11, No. 05, pp. 23-30, 1994.

- (Lucrédio et al., 2007) Lucrédio, D.; Brito, K.S.; Alvaro, A.; Garcia, V.C.; Almeida, E.S.; Fortes, R.P.M.; Meira, S.R.L. **Software Reuse: The Brazilian Industry Scenario**, In: *Journal of Systems and Software*, Elsevier, 2007.
- (McCall et al., 1977) McCall, J.A; Richards, P.K.; Walters, G.F. **Factors in Software Quality**, *Griffiths Air Base, Nova York, Rome Air Development Center (RADC) System Command, TR-77-369*, Vol. I, II and III, 1977.
- (McGarry et al., 2002) McGarry, J.; Card, D.; Jones, C.; Layman, B.; Clark, E.; Dean, J.; Hall, F., **Practical Software Measurement: Objective Information for Decision Makers**, ISBN 0-201-71516-3, Addison-Wesley, 2002.
- (McGregor et al., 2003) McGregor, J. D.; Stafford, J. A.; Cho, I. H. **Measuring Component Reliability**, In: *The 6th Workshop on Component-Based Software Engineering (CBSE)*, Lecture Notes in Computer Science (LNCS) Springer-Verlag, USA, pp. 13-24, 2003.
- (McIlroy, 1968) McIlroy, M. D. **Mass Produced Software Components**, In: *NATO Software Engineering Conference Report*, Garmisch, Germany, pp. 79-85, 1968.
- (Merrit, 1994) Merrit, S. **Reuse Library**, In: *Encyclopedia of Software Engineering*, J.J. Marciniak (editor), John Willey & Sons, pp. 1069-1071, 1994.
- (Meyer, 2003) Meyer, B. **The Grand Challenge of Trusted Components**, In: *The 25th IEEE International Conference on Software Engineering (ICSE)*, USA, pp. 660–667, 2003.
- (Meyer, 1999) Meyer, B.; Mingins, C. **Component-Based Development: From Buzz to Spark**, In: *IEEE Computer*, Vol. 32, No. 07, pp. 35-37, 1999.
- (Meyer, 1997) Meyer, B. **Object-Oriented Software Construction**, *Prentice Hall*, 2th Edition, London, 1997.

- (Microsoft COM, 2007) Microsoft COM Technologies, At <http://www.microsoft.com/com>. Consulted in August 2007.
- (Mili et al., 1998) Mili, A.; Mili, R.; Mittermeir, R. T. **A Survey of Software Reuse Libraries**, In: *Annals Software Engineering*, Vol. 05, No. 01, pp. 349–414, 1998.
- (Mingins et al., 1998) Mingins, C., Schmidt, H., **Providing Trusted Components to the Industry**. In: *IEEE Computer*, Vol. 31, No. 05, pp. 104-105, 1998.
- (Mahmooda et al, 2005) Mahmooda, S.; Laia, R.; Kimb, Y.S.; Kimb, J.H.; Parkb, S.C.; Ohb, H.S. **A survey of component based system quality assurance and assessment**, In: *Journal of Information and Software Technology*, Vol. 47, No. 10, pp. 693-707, 2005.
- (Mohagheghi and Conradi, 2007) Mohagheghi, P.; Conradi, R. **Quality, productivity and economic benefits of software reuse: a review of industrial studies**, *Empirical Software Engineering*, Vol. 12, No. 05, pp. 471-516, 2007.
- (Morisio et al., 2002) Morisio, M.; Ezran, M.; Tully, C. **Success and Failure Factors in Software Reuse**, In: *IEEE Transactions on Software Engineering*, Vol. 28, No. 04, pp. 340-357, 2002.
- (Morris et al., 2001) Morris, J.; Lee, G.; Parker, K.; Bundell, G. A.; Lam, C. P.; **Software Component Certification**. In: *IEEE Computer*, Vol. 34, No. 09, pp. 30-36, 2001.
- (Oliveira et al., 2007) Oliveira, R.Y.S.; Ferreira, P.G.; Alvaro, A.; Almeida, E.S.; Meira, S. L. **Code Inspetion: A Review**, In: *The 9th International Conference on Enterprise Information Systems (ICEIS)*, Poster Session, Madeira, Portugal. Lecture Notes in Computer Science (LNCS), Springer-Verlag. 2007.
- (OMG, 2007) Object Management Group (OMG), At <http://www.omg.org>. Consulted in August 2007.

- (OMG CCM, 2002) Object Management Group (OMG), **CORBA Components**, Version 3, Document num. formal/02-06-65, June 2002.
- (Parnas and Lawford, 2003) Parnas, D.; Lawford, M. **The Role of Inspection in Software Quality Assurance**. In: *IEEE Transactions on Software Engineering*, Vol. 29, No. 08, pp. 674-676, 2003.
- (Poore et al., 1993) Poore, J.; Mills, H.; Mutchler, D. **Planning and Certifying Software System Reliability**, In: *IEEE Computer*, Vol. 10, No. 01, pp. 88-99, 1993.
- (Pressman, 2005) Pressman, R. **Software Engineering: A Practitioner's Approach**. McGraw-Hill. 6th Edition. 2005.
- (Prieto-Díaz, 1990) Prieto-Díaz, R. **Domain analysis: an introduction**, In: *ACM SIGSOFT Software Engineering Notes*, Vol. 15, No. 02, pp. 47-54, 1990.
- (Rada et al., 1997) Rada, R.; Moore, J. W. **Standardizing Reuse**, In: *Communications of the ACM*, Vol. 40, No. 03, pp. 19-23, 1997.
- (Ramos et al., 2007a) Ramos, R.A.; Silva, J.; Alvaro, A.; Afonso, R.A. **PHP for Professionals (in portugues)**, Editora Digerati, ISBN: 978-85-60480-64-7, 2007.
- (Ramos et al., 2007b) Ramos, R.A.; Silva, J.; Alvaro, A.; **Essencial de VBA** Editora, Digerati, ISBN: 978-85-60480-67-8, 2007.
- (Reussner, 2003) Reussner, R. H. **Contracts and quality attributes of software components**, In: *The 8th International Workshop on Component-Oriented Programming (WCOP) in conjunction with the 17th ACM European Conference on Object Oriented Programming (ECCOP)*, 2003.
- (Rohde et al., 1996) Rohde, S. L.; Dyson, K. A.; Geriner, P. T.; Cerino, D. A. **Certification of Reusable Software Components: Summary of Work in Progress**, In: *The 2nd IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, Canada, pp. 120-123, 1996.

- (Roos, 1997) Ross, D.T. **Structured Analysis (SA): A Language for Communicating Ideas**, In: *IEEE Transaction on Software Engineering*, Vol. 03, No. 01, pp. 16-34, 1997.
- (Sametinger, 1997) Sametinger, J. **Software Engineering with Reusable Components**. Springer Verlag, USA, 1997.
- (Schmidt, 2003) Schmidt, H. **Trustworthy components: compositionality and prediction**. In: *Journal of Systems and Software*, Vol. 65, No. 03, pp. 215-225, 2003.
- (Schneider & Han, 2004) Schneider, J-G.; Han, J. **Components — the Past, the Present, and the Future**, In: *Proceedings of Ninth International Workshop on Component-Oriented Programming (WCOP)*, Oslo, Norway, June 2004.
- (Shukla et al., 2004) Shukla, R.Y.; Strooper, P.A.; Carrington, D.A. **A Framework for Reliability Assessment of Software Components**, In: *Proceedings of the 7th International Symposium on Component-based Software Engineering (CBSE)*, Edinburgh, UK, pp. 272-279, 2004.
- (Simão et al., 2003) Simão, R. P. S.; Belchior, A. **Quality Characteristics for Software Components: Hierarchy and Quality Guides, Component-Based Software Quality: Methods and Techniques**, In: *Lecture Notes in Computer Science (LNCS)* Springer-Verlag, Vol. 2693, pp. 188-211, 2003.
- (Slaughter et al., 1998) Slaughter, S.; Harter, D.; Krishnan, M. **Evaluating the Cost of Software Quality**, In: *Communications of the ACM*, Vol. 31, No.08, pp 67-73, 1998.
- (Softex, 2007) Softex, **Development Perspectives and Components Usage in the Brazilian Software Industry (in portuguese)** Campinas: SOFTEX, 2007. Available at: [http://golden.softex.br/portal/softexweb/uploadDocuments/Compone ntes\(2\).pdf](http://golden.softex.br/portal/softexweb/uploadDocuments/Compone ntes(2).pdf)

- (Solingen, 2000) Solingen, R.v. **Product focused software process improvement: SPI in the embedded software domain**, *PhD Thesis*, Technische Universiteit Eindhoven, 2000.
- (Stafford et al., 2001) Stafford, J.; Wallnau, K. C. **Is Third Party Certification Necessary?**, In: *The 4th Workshop on Component-Based Software Engineering (CBSE)*, Lecture Notes in Computer Science (LNCS) Springer-Verlag, Canada, 2001.
- (Szyperski, 2002) Szyperski, C. **Component Software: Beyond Object-Oriented Programming**. Addison-Wesley, USA, 2002.
- (Taulavuori et al., 2004) Taulavuori, A.; Niemela, E.; Kallio, P. **Component documentation — a key issue in software product lines**, In: *Journal Information and Software Technology*, Vol. 46, No. 08, June, pp. 535–546, 2004.
- (Tian, 2004) Tian, J. **Quality-Evaluation Models and Measurements**, In: *IEEE Software*, Vol. 21, No.03, pp.84-91, May/June, 2004.
- (TMMi, 2008) TMMi Foundation, **Test Maturity Model Foundation (TMMi)**, Version 1.0, February, 2008.
- (Trass et al., 2000) Trass, V.; Hillegersberg, J. **The software component market on the Internet, current status and conditions for growth**, In: *ACM Sigsoft Software Engineering Notes*, Vol. 25, No. 01, pp. 114-117, 2000.
- (Vitharana, 2003) Vitharana, P. **Risks and Challenges of Component-Based Software Development**, In: *Communications of the ACM*, Vol. 46, No. 08, pp.67-72, 2003.
- (Voas et al., 2000) Voas, J. M.; Payne, J. **Dependability Certification of Software Components**, In: *Journal of Systems and Software*, Vol. 52, No. 2-3 , pp. 165-172, 2000.
- (Voas, 1998) Voas, J. M. **Certifying Off-the-Shelf Software Components**, In: *IEEE Computer*, Vol. 31, No. 06, pp. 53-59, 1998.
- (Wallin, 2002) Wallin, C. **Verification and Validation of Software Components and Component Based Software Systems**, In:

- Building Reliable Component-Based Systems*, I. Crnkovic, M. Larsson (editors), Artech House Publishers, July, pp. 29-37, 2002.
- (Wallnau, 2004) Wallnau, K., **Software Component Certification: 10 Useful Distinctions**, In: *Technical Note CMU/SEI-2004-TN-031*, 2004. Available at: <http://www.sei.cmu.edu/publications/documents/04.reports/04tn031.html>.
- (Wallnau, 2003) Wallnau, K. C. **Volume III: A Technology for Predictable Assembly from Certifiable Components**. In: *Software Engineering Institute (SEI), Technical Report*, Vol. III, April, 2003.
- (Washizaki et al., 2003) Washizaki, H.; Yamamoto, H.; Fukazawa, Y. **A metrics suite for measuring reusability of software components**, In: *Proceedings of the 9th IEEE International Symposium on Software Metrics*, pp.211–223, 2003.
- (Weber et al., 2002) Weber, K. C.; Nascimento, C. J.; **Brazilian Software Quality 2002**. In: *The 24th IEEE International Conference on Software Engineering (ICSE)*, EUA, pp. 634-638, 2002.
- (Wohlin et al., 2000) Wohlin, C.; Runeson, P.; Host, M.; Ohlsson, C.; Regnell, B.; Wesslén, A. **Experimentation in Software Engineering: an Introduction**, Kluwer Academic Publishers, Norwell, 2000.
- (Wohlin et al., 1994) Wohlin, C.; Runeson, P. **Certification of Software Components**, In: *IEEE Transactions on Software Engineering*, Vol. 20, No. 06, pp 494-499, 1994.
- (Wohlin and Regnell, 1998) Wohlin, C.; Regnell, B. **Reliability Certification of Software Components**, In: *The 5th IEEE International Conference on Software Reuse (ICSR)*, Canada, pp. 56-65, 1998.
- (Woodman et al., 2001) Woodman, M.; Benebiktsson, O.; Lefever, B.; Stallinger, F. **Issues of CBD Product Quality and Process Quality**, In: *The 4th Workshop on Component-Based Software Engineering (CBSE)*, Lecture Notes in Computer Science (LNCS), Springer-Verlag, Canada, 2001.

Appendix A. Metrics Example

Chapter 8 presented some brief examples of metrics definition using the GQM approach. In order to help the evaluation team during the metrics definition, now some other related metrics that could be considered in a component evaluation will be described. However, the more complex the sub-characteristics and its related quality attribute are, the more difficult it is to provide metrics examples without a well-defined context.

- **Example of Metrics to track Functionality Characteristics**

For Functionality characteristics there are five sub-characteristics that could be evaluated: Accuracy, Security, Suitability, Interoperability, Compliance and Self-Contained. These have a set of quality attributes, for which will be presented at least one metric as example of usage, as follows.

For the **Accuracy Sub-Characteristic** the following metrics could be applied:

Functionality	
Sub-Characteristic	Accuracy
Quality Attribute	Correctness
Goal	Evaluates the percentage of the results that were obtained with precision
Question	Based on the amount of tests executed, how much test results return with precision?
Metric	Precision on results / Amount of tests
Interpretation	$0 \leq x \leq 1$; closer to 1 being better

For the **Security Sub-Characteristic**, the following metrics could be applied:

Functionality	
Sub-Characteristic	Security
Quality Attribute	Data Encryption
Goal	Evaluate the encryption of the input and output data of the component.
Question	How complete is the data encryption implementation?
Metric	Number of services that must have data encryption / Number of services that have encryption
Interpretation	$0 \leq x \leq 1$; closer to 1 being better

Functionality	
Sub-Characteristic	Security
Quality Attribute	Controllability
Goal	Evaluate if the component provide any control mechanism.
Question	How controllable is the component access?
Metric	Number of provided interfaces that control the access / Number of provided interfaces
Interpretation	$0 \leq x \leq 1$; closer to 1 being better

Functionality	
Sub-Characteristic	Security
Quality Attribute	Auditability
Goal	Evaluate if the component provide any audit mechanism.
Question	How controllable is the component audit mechanism?
Metric	Number of provided interfaces that log-in the access (or any kind of data) / Number of provided interfaces
Interpretation	$0 \leq x \leq 1$; closer to 1 being better

For the **Suitability Sub-Characteristic** the following metrics could be applied:

Functionality	
Sub-Characteristic	Suitability
Quality Attribute	Coverage
Goal	Evaluates the implementation coverage.
Question	How much of the required functions is

	covered by the component implementation?
Metric	% of functions implemented / Specified functions
Interpretation	$0 \leq x \leq 1$; closer to 1 being better

Functionality	
Sub-Characteristic	Suitability
Quality Attribute	Completeness
Goal	Evaluates the completeness of each implemented function
Question	How much of the implemented functions are totally implemented?
Metric	Total of specified functions / Implemented functions (complete)
Interpretation	$0 \leq x \leq 1$; closer to 1 being better

Functionality	
Sub-Characteristic	Suitability
Quality Attribute	Pre and Post-Conditioned
Goal	Evaluates the ability of the component to provide the pre and post conditions
Question	How many of the provided and required interfaces contain pre and post conditions?
Metric	Total provided and required interfaces / number of interfaces with pre and post-conditions
Interpretation	$0 \leq x \leq 1$; closer to 1 being better

Functionality	
Sub-Characteristic	Suitability
Quality Attribute	Proofs of Pre and Post-Conditions
Goal	Evaluates the ability of the component to prove the pre and post conditions
Question	How much of pre and post-conditions are formally proved?
Metric	Number of interfaces with pre and post-conditions / Number of formally proved interfaces
Interpretation	$0 \leq x \leq 1$; closer to 1 being better

For the **Interoperability Sub-Characteristic** the following metrics could be applied:

Functionality	
Sub-Characteristic	Interoperability
Quality Attribute	Data Compatibility

Goal	Evaluates the compatibility of the component's data to any international standard
Question	How correctly were implemented the data standard?
Metric	Number of provided interfaces using any data standard in a correct way / Number of provided interfaces
Interpretation	$0 \leq x \leq 1$; closer to 1 being better

For the **Compliance Sub-Characteristic** the following metrics could be applied:

Functionality	
Sub-Characteristic	Compliance
Quality Attribute	Standardization
Goal	Evaluates the component standards, if there is any.
Question	How many standards are used/provided in the component?
Metric	Number of functions using some standard / Number of functions
Interpretation	$0 \leq x \leq 1$; closer to 1 being better

Functionality	
Sub-Characteristic	Compliance
Quality Attribute	Certification
Goal	Evaluates the ability to provide any certified functions
Question	How many functions were certified by any recognized organization?
Metric	Number of functions certified / Total number of functions
Interpretation	$0 \leq x \leq 1$; closer to 1 being better

For the **Self-Contained Sub-Characteristic** the following metrics could be applied:

Functionality	
Sub-Characteristic	Self-Contained
Quality Attribute	Dependability
Goal	Evaluates the ability of the component to provide itself all functions expected
Question	How many functions does the component provide by itself?
Metric	Number of functions provided by itself / Number of specified functions

Interpretation	$0 \leq x \leq 1$; closer to 1 being better
-----------------------	--

• Example of Metrics to track Reliability Characteristics

For Reliability characteristic there are three sub-characteristics that could be evaluated: Fault Tolerance, Recoverability and Maturity. These have a set of quality attributes, for which will be presented at least one metric as example of usage, as follows.

Besides those that are presented here, some reliability metrics for software components found on literature can be considered (McGregor et al., 2003), (Shukla et al., 2004).

For the **Fault Tolerance Sub-Characteristic**, the following metrics could be applied;

Reliability	
Sub-Characteristic	Fault Tolerance
Quality Attribute	Mechanism available
Goal	Evaluates the functions that contain fault tolerance mechanism
Question	How many functions provide the fault tolerance mechanism?
Metric	Number of functions that contain any kind of fault tolerance mechanism / Number of functions
Interpretation	$0 \leq x \leq 1$; closer to 1 being better

Reliability	
Sub-Characteristic	Fault Tolerance
Quality Attribute	Mechanism Efficiency
Goal	Evaluates the efficiency of the fault tolerance mechanism
Question	<ul style="list-style-type: none"> - How is the efficiency of the functions that provide any kind of fault tolerance mechanism? - What it is the range of the data lost?
Metric	<ul style="list-style-type: none"> - Number of functions that contain any kind of fault tolerance mechanism / Number of mechanisms that are considered efficient - Total number of interfaces that exchanged data to outside / Number of interface that lost data
Interpretation	$0 \leq x \leq 1$; closer to 1 being better

For the **Recoverability Sub-Characteristic** the following metrics could be applied:

Reliability	
Sub-Characteristic	Recoverability
Quality Attribute	Error Handling
Goal	Evaluates the ability of the component to avoid error situations.
Question	How many functions provide any mechanism that can avoid error situations?
Metric	Number of error handlings provided / Number of functions
Interpretation	$0 \leq x \leq 1$; closer to 1 being better

For the **Maturity Sub-Characteristic** the following metrics could be applied:

Reliability	
Sub-Characteristic	Maturity
Quality Attribute	Volatility
Goal	Analyzes the average time between commercial versions.
Question	What is the average time between the component versions?
Metric	Number of versions / Time spent to release a new version (in days)
Interpretation	$0 \leq x \leq 1$; closer to 1 being better * If the result is more than 1 it means that the vendor constantly improve the component functions and should be considered as the maximum degree of maturity.

Reliability	
Sub-Characteristic	Maturity
Quality Attribute	Failure Removal
Goal	Analyzes the amount of failure removal per component version
Question	How many bugs were corrected during the component versions?
Metric	Number of versions / Number of bugs corrected per version
Interpretation	$0 \leq x \leq 1$; which closer to 0 being better

- **Example of Metrics to track Usability Characteristics**

For the Usability characteristic, there are four sub-characteristics that could be evaluated: Configurability, Understandability, Learnability and Operability. These have a set of quality attributes, for which will be presented at least one metric as example of usage, as follows.

Besides the ones presented here, some usability metrics for software components found on literature can be considered (Bertoa & Vallecillo, 2004), (Bertoa et al., 2006).

For the **Configurability Sub-Characteristic** the following metrics could be applied:

Usability	
Sub-Characteristic	Configurability
Quality Attribute	Effort to Configure
Goal	Evaluates the time necessary to configure the component.
Question	How much time is needed to configure the component in order to work correctly in a system?
Metric	Time spent to configure correctly
Interpretation	The faster it is to configure the component the better, but it depends of the component and environment complexity.

For the **Understandability Sub-Characteristic** the following metrics could be applied:

Usability	
Sub-Characteristic	Understandability
Quality Attribute	Document available
Goal	Analyses the documentation availability.
Question	How many documents are available to understand the component functions?
Metric	Number of documents
Interpretation	The higher number of documents available is better, but it depends of the component complexity, domain, etc.

Usability	
Sub-Characteristic	Understandability
Quality Attribute	Document readability and quality
Goal	Analyses the efficiency and efficacy of the provided documents.

Question	How is the quality of the provided documents?
Metric	Amount of documents with quality / Amount of documents provided
Interpretation	$0 \leq x \leq 1$; closer to 1 being better

Usability	
Sub-Characteristic	Understandability
Quality Attribute	Code Readability
Goal	Analyzes the source code
Question	How easy it is to understand the source code?
Metric	Time spent to understand the source code
Interpretation	Based on the Number of Lines of Code (LOC); analysis of the modularity, coupling, cohesion and simplicity.

For the **Learnability Sub-Characteristic** the following metrics could be applied:

Usability	
Sub-Characteristic	Learnability
Quality Attribute	Time and effort to (use, configure, admin and expertise) the component
Goal	Evaluates the effort necessary to use, configure, admin and expertise in the component.
Question	How much effort is needed to (use, configure, admin and expertise) the component?
Metric	Time spend to learn the component abilities
Interpretation	The faster to learn the better, but depends on the component complexity

For the **Operability Sub-Characteristic** the following metrics could be applied:

Usability	
Sub-Characteristic	Operability
Quality Attribute	Complexity level
Goal	Analyzes the ability to operate all provided functions
Question	How much time it is needed to operate the component?
Metric	All functions usage / time to operate

Interpretation	The lower the better (Σ usage time of each function)
-----------------------	--

Usability	
Sub-Characteristic	Operability
Quality Attribute	Provided Interfaces
Goal	Analyses the complexity of the provided interfaces
Question	How many functions and parameters are necessary to execute the component functions?
Metric	Number of provided functions and parameters (NPFP) * N / Number of provided interfaces Where N can assume the following value: <ul style="list-style-type: none"> - NPFP < 4, N=1 - 4 <= NPFP <= 6, N=2 - NPFP >=7, N=3
Interpretation	0 <= x <= 1; closer to 1 being better

Usability	
Sub-Characteristic	Operability
Quality Attribute	Required Interfaces
Goal	Analyses the complexity of the required interfaces
Question	How many functions and parameters are necessary to execute the component functions?
Metric	Number of required functions and parameters * N / Number of required interfaces Where N can assume the following value: <ul style="list-style-type: none"> - <4, N=1 - >=4<=6, N=2 - >=7, N=3
Interpretation	0 <= x <= 1; closer to 1 being better

Usability	
Sub-Characteristic	Operability
Quality Attribute	Effort for Operating
Goal	Analyses the complexity to operate the functions provided by the component
Question	How many operations are provided by each interface?
Metric	Number of operations in all provided interfaces / Number of provided interfaces
Interpretation	0 <= x <= 1; closer to 1 being better

- **Example of Metrics to track Efficiency Characteristics**

For the Efficiency characteristic there are three sub-characteristics that could be evaluated: Time Behavior, Resource Behavior and Scalability. These have a set of quality attributes, for which will be presented at least one metric as example of usage, as follows.

For the **Time Behavior Sub-Characteristic** the following metrics could be applied:

Efficiency	
Sub-Characteristic	Time Behavior
Quality Attribute	Response Time
Goal	Evaluates the time taken since a request is received until a response has been sent
Question	How is the average time between the response times?
Metric	$(\sum \text{Time taken between a set of invocations per each provided interface}) / \text{Number of invocations}$
Interpretation	$0 \leq x \leq 100$; which closer to 100 being better

Efficiency	
Sub-Characteristic	Time Behavior
Quality Attribute	Latency – Throughput
Goal	Analyses the amount of output that can be successfully produced over a given period of time.
Question	How much output can be produced with success over a period of time?
Metric	$(\text{Amount of output with success over a period of time} * 100) / \text{Number of invocations}$
Interpretation	$0 \leq x \leq 100$; which closer to 100 being better

Efficiency	
Sub-Characteristic	Time Behavior
Quality Attribute	Latency – Processing Capacity
Goal	Analyses the amount of input information that can be successfully processed by the component over a given period of time
Question	How much input can be processed with success over a period of time?

Metric	(Amount of input processed with success over a period of time * 100) / Number of invocations
Interpretation	$0 \leq x \leq 100$; which closer to 100 being better

For the **Resource Behavior Sub-Characteristic** the following metrics could be applied:

Efficiency	
Sub-Characteristic	Resource Behavior
Quality Attribute	Memory Utilization
Goal	Analyzes the amount of memory required to its correct work.
Question	How much memory is enough for the component to work correctly?
Metric	Amount of memory necessary for the component to work correctly/amount of memory available on the execution environment.
Interpretation	$0 \leq x \leq 1$; which closer to 0 being better

Efficiency	
Sub-Characteristic	Resource Behavior
Quality Attribute	Disk Utilization
Goal	Analyzes the amount of disk space required to its correct work.
Question	How much disk space is enough for the component to work correctly?
Metric	Amount of disk necessary for the component to work correctly/amount of disk available on the execution environment.
Interpretation	$0 \leq x \leq 1$; which closer to 0 being better

For the **Scalability Sub-Characteristic** the following metrics could be applied:

Efficiency	
Sub-Characteristic	Scalability
Quality Attribute	Processing Capacity
Goal	Evaluates the capacity of the component to support a huge volume of data.
Question	How does the component responds with increase of data being processed?
Metric	$(\sum \text{response time of each call}) / \text{Number of calls}$ * This can be executed in a defined time

	and the number of calls could vary. Thus the average ratio between the calls can be calculated.
Interpretation	It depends on the time defined to be executed. It should be analyzed based on the average calls and, before, define the period of time to execute this measurement.

• Example of Metrics to track Maintainability Characteristics

For the Maintainability characteristic there are three sub-characteristics that could be evaluated: Stability, Changeability and Testability. These have a set of quality attributes, for which will be presented at least one metric as example of usage, as follows.

For the **Stability Sub-Characteristic** the following metrics could be applied:

Maintainability	
Sub-Characteristic	Stability
Quality Attribute	Modifiability
Goal	Evaluates the flexibility to change the component source code in order to improve its functions
Question	How modifiable is the component?
Metric	Execute a set of modifications and analyze the component behavior
Interpretation	Analyze the amount of modifications done and the amount of modifications that works well

For the **Changeability Sub-Characteristic** the following metrics could be applied:

Maintainability	
Sub-Characteristic	Changeability
Quality Attribute	Extensibility
Goal	Evaluates the flexibility to extend the component functions
Question	How extensible is the component?
Metric	Execute a set of extensions and analyze the new component behavior
Interpretation	Analyze the amount of extensions done and the amount of extensions that work well

Maintainability	
Sub-Characteristic	Changeability
Quality Attribute	Costumizability
Goal	Analyzes the customizable parameters that the component offers
Question	How much parameters are provided to customize each function of the component?
Metric	Number of provided interfaces / Number of parameters to configure the provided interface
Interpretation	$0 \leq x \leq 1$; which closer to 1 is better

Maintainability	
Sub-Characteristic	Changeability
Quality Attribute	Modularity
Goal	Analyzes the internal organization of the component
Question	How logically separated are the component concerns?
Metric	Packaging analysis
Interpretation	If the component contains some packages that isolate each logical concern it probably has good modularity. On the other hand, if the component doesn't contain a well defined internal structure the modularity level is slower.

For the **Testability Sub-Characteristic** the following metrics could be applied:

Maintainability	
Sub-Characteristic	Testability
Quality Attribute	Test Suite Provided
Goal	Analyzes the ability of the component to provide some test suite for checking its functions
Question	<ul style="list-style-type: none">- Is there any test suite?- How is the coverage of this test suite based on the whole component functions?
Metric	<ul style="list-style-type: none">- Analysis of the test suites provided- Number of test suites provided / Number of functions
Interpretation	$0 \leq x \leq 1$; closer to 1 being better

Maintainability	
Sub-Characteristic	Testability
Quality Attribute	Extensive Component Test Case
Goal	Analyzes if the component was extensively tested before being made available to the market
Question	<ul style="list-style-type: none"> - How many tests cases are executed? - What is the coverage of these test cases?
Metric	Number of functions / Number of test cases * Still on it is interesting to analyze the number of bugs that were corrected during the test case
Interpretation	The test cases coverage is very important to be analyzed and the number of bugs discovered during the execution of the tests.

Maintainability	
Sub-Characteristic	Testability
Quality Attribute	Component Test in a Specific Environment
Goal	Analyzes the environments where the component can work well
Question	In which environment this component can be executed without errors?
Metric	Number of environments that work well / Number of environments defined on specification
Interpretation	$0 \leq x \leq 1$; closer to 1 being better

Maintainability	
Sub-Characteristic	Testability
Quality Attribute	Proofs the Components
Goal	Analyzes if the tests are formally proved
Question	How is the coverage of the proof in the test cases?
Metric	Proofs Analysis
Interpretation	It is interesting to note if the amount of formal proof covers the whole test cases provided by the component. As higher it is better.

• Example of Metrics to track Portability Characteristics

For the Portability there are five sub-characteristics that could be evaluated: Deployability, Replaceability, Adaptability and Reusability. These

have a set of quality attributes, for which will be presented at least one metric as example of usage, as follows.

For the **Deployability Sub-Characteristic** the following metrics could be applied:

Portability	
Sub-Characteristic	Deployability
Quality Attribute	Complexity Level
Goal	Analyzes how complex it is to deploy a component in its specific environment(s)
Question	How much time does it take to deploy a component in its environment?
Metric	Time taken for deploying a component in its environment
Interpretation	Estimate the time first and then compare with the actual time taken to deploy the component

For the **Replaceability Sub-Characteristic** the following metrics could be applied:

Portability	
Sub-Characteristic	Replaceability
Quality Attribute	Backward Compatibility
Goal	Analyzes the compatibility with previous versions
Question	What is the compatibility with previous versions?
Metric	Correct results / Set of same invocations in different component versions
Interpretation	$0 \leq x \leq 1$; closer to 1 being better

For the **Adaptability Sub-Characteristic** the following metrics could be applied:

Portability	
Sub-Characteristic	Adaptability
Quality Attribute	Mobility
Goal	Analyzes the ability of the component to be transferred from one environment to another
Question	Can the component be transferred to other environment without any changes?
Metric	<ul style="list-style-type: none"> - Analyze the component constraints and environment constraints - Analyze the component specification

	<ul style="list-style-type: none"> - Deploy the component in environment specified on documentation <p>* Possible metric: Number of environments where the component works correctly / Number of environments described in its specification</p>
Interpretation	$0 \leq x \leq 1$; closer to 1 being better

Portability	
Sub-Characteristic	Adaptability
Quality Attribute	Configuration Capacity
Goal	Analyzes the ability of the component to be transferred from one environment to another, considering the related changes
Question	How much effort is needed to adapt the component to a new one environment?
Metric	<ul style="list-style-type: none"> - Analyze the component constraints and environment constraints - Analyze the component specification - Deploy the component in environment specified on documentation - Time taken to adapt the component in its specified environments
Interpretation	Analyze the time taken to deploy the component in each environment defined

For the **Reusability Sub-Characteristic** the following metrics could be applied. It could also be considered other reusability metrics for software components found on literature (Washizaki et al., 2003), (Bay and Pauls, 2004).

Portability	
Sub-Characteristic	Reusability
Quality Attribute	Domain Abstraction Level
Goal	Analyzes the correct separation of concerns in the component
Question	<ul style="list-style-type: none"> - Can the component be reused in other domain applications? - Does the component have inter-related business code?
Metric	Analyzes the source code and tries to reuse the component in other domains
Interpretation	If the component does not contain business code related to specific domain and can be reused around a set of domains, it is good candidate to be

	reused. On the other hand, if it does have code related to a specific domain and it becomes difficult to reuse it around some domain, the component is not good candidate to be reusable and should be revised.
--	---

Portability	
Sub-Characteristic	Reusability
Quality Attribute	Architecture Compatibility
Goal	Analyzes the level of dependability of a specified architecture
Question	Was the component correctly designed based on the architecture constraints defined?
Metric	Analysis of the component design based on some documentation and source code
Interpretation	Understand the architecture constraints and analyze if the component follows that one specification during its development and implementation. Based on this, the component can be considered good to be reused or not.

Portability	
Sub-Characteristic	Reusability
Quality Attribute	Modularity
Goal	Analyzes the internal organization of the component
Question	How logically separated are the component concerns?
Metric	Packaging analysis
Interpretation	If the component contains some packages that isolate each logical concern, it should have good modularity and become more reusable and extensible. On the other hand, if the component doesn't contain a well define internal structure, the modularity level is slower and, consecutively, the reusability level decreases.

Portability	
Sub-Characteristic	Reusability
Quality Attribute	Cohesion
Goal	Analyzes the cohesion between the internal modules/packages/functionalities of the component

Question	How is the cohesion level of the component?
Metric	Analysis of the inter-related parts
Interpretation	A component should have high cohesiveness in order to increase its reusability level;

Portability	
Sub-Characteristic	Reusability
Quality Attribute	Coupling
Goal	Analyzes the coupling between the internal modules/packages/functionalities of the component
Question	How is the coupling level of the component?
Metric	Analysis of the inter-related parts (Call-Called modules, methods, etc)
Interpretation	A component should have low coupling in order to increase its reusability level

Portability	
Sub-Characteristic	Reusability
Quality Attribute	Simplicity
Goal	Analyzes the way the component is organized
Question	How simple is the component's organization?
Metric	Number of modules, average module size and cyclomatic complexity
Interpretation	The component's organization should be easily understandable and (re)usable. The simpler the better.

• Metrics to track the Evaluation Techniques Properties

With the same objective of the metrics cited above, now some metrics to track the properties of the evaluation techniques proposed in Chapter 6 will be described. Differently from the last sections, where one metric was presented for each quality attribute, now, some just examples of metrics to track the properties of the evaluation techniques defined on Software Component Techniques Model (SCTM) will be presented because each technique can be measured in different ways and complexity, using different tools, techniques, methods and processes. Thus, the evaluation team should define which degree of thoroughness it is more interesting to measure each evaluation technique

proposed. Some recognized tools or methods from the literature will be used as basis (considering that the evaluated components were developed using Java programming language), as follows:

Functionality	
Quality Attribute	Response Time
SCTM level	I
Technique	Accuracy Analysis using TPTP tool ¹⁷
Goal	Evaluates the percentage of the time taken between a set of invocations
Question	Is the tool efficient to measure the response time of this kind of component? If so, how accurate are the output results from the component?
Metric	Analysis of the results and coverage of the tool
Interpretation	<p>Definition of the applicability of the tool to measure these quality attributes. If this tool can efficiently measure the response time of a set of invocations, it is good. On the other hand, if it is not enough to evaluate some functions, other tools should be used to complement or to substitute this one.</p> <p>If this tool is good to evaluate the component, an analysis of how many results are generated with the expected accuracy could include the following formula:</p> $\frac{\text{Number of results with accuracy}}{\text{Number of results generated}}$ <p>0 <= x <= 1; closer to 1 being better</p>

Efficiency	
Quality Attribute	Processing Capacity, Memory and Disk utilization
SCTM level	III
Technique	Test of Performance using JMeter tool ¹⁸
Goal	Evaluates the processing capacity of the component together with its memory and disk usage
Question	Can JMeter evaluate in an efficient way the performance of this component?
Metric	Based on the component knowledge, analyze if the tool can support/evaluate all functionality performance

¹⁷ Eclipse Test & Performance Tools Platform Project (TPTP) – <http://www.eclipse.org/tptp>

¹⁸ Apache JMeter – <http://jakarta.apache.org/jmeter>

Interpretation	<p>If the tool can evaluate in an efficient way the performance of each functions through its resources, it is a good tool to use during evaluation. On the other hand, if it is not enough to evaluate some kind of performance, other tool should be use to complement or to substitute this one.</p> <p>If this tool is good to evaluate the component, an analysis of how much memory and disk is necessary and the processing capacity of the component could include the following formula:</p> $\text{(Number of outputs with success over a period of time * 100) / Number of invocations}$ <p>$0 \leq x \leq 100$; which closer to 100 being better</p>
-----------------------	---

Portability	
Quality Attribute	Coupling, Cohesion, Simplicity, Reusability and Modularity analyzes
SCTM level	I
Technique	Coupling, Cohesion, Simplicity, Reusability, Modularity analyzes using Checkstyle tool ¹⁹
Goal	Evaluates the internal source code of the component
Question	Is the Checkstyle tool efficient enough to measure those attributes?
Metric	Analysis of the results and coverage of the tool
Interpretation	<p>If the tool can mine these kinds of information from the source code and present them to be analyzed, it is good to evaluate the component. On the other hand, if it is not enough to evaluate some kind of attributes, other tool should be use to complement or to substitute this one.</p> <p>If this tool is good to evaluate the component, an analysis of the metrics collected in the tool can be used to define those attributes from the component.</p> <p>The idea is that the component should have: less coupling, high cohesion, high modularity, ways to perform the function in a simple way and high reusability.</p>

¹⁹ Checkstyle – <http://checkstyle.sourceforge.net>

- **Metrics to track the Evaluation Process Proprieties**

Consistent and good evaluation results can only be achieved by following a high quality and consistent evaluation process (Comella-Dorda et al., 2003). However, to assure the efficiency and efficacy of the process it is important to define some metrics. The idea is to obtain feedback from those metrics in order to improve the activities and steps proposed to evaluate the component quality (that were presented on Chapter 7). Next, two metrics that could be used with this purpose will be presented:

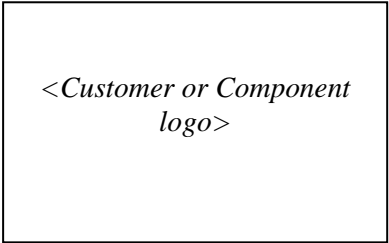
Component Evaluation Process	
Goal	Adequately evaluate the software component.
Question	Can the evaluation team evaluate everything they planned to execute using the techniques, documents and models developed during the process activities?
Metric	Total documented functions / Total component functions (or Total measurement accomplished)
Interpretation	$0 \leq x \leq 1$; closer to 1 being better.

Component Evaluation Process	
Goal	Analyzes the usability of the templates provided
Question	Did the templates helped during the evaluation development?
Metric	Evaluation team feedback
Interpretation	If the templates did not help during the evaluation development, they should be adapted to improve the time of the component evaluation process. Number of positive feedbacks / All feedbacks $0 \leq x \leq 1$; closer to 1 being better.

Appendix B. Component Quality Evaluation Form

Component Certification

<Component Name>



Version <Document Version> | <Document date version>

Responsible: <Responsible Name>



Historic Changes

Date	Version	Description	Author

Contents

1.	Introduction	192
1.1	Overview of the Component	192
1.2	Conventions, terms and abbreviations list	192
2.	Software Component Evaluation	193
2.1	Establish Evaluation Requirements activity	193
2.1.1	Form an Evaluation Team	193
2.1.2	Define the Goals and Scope	193
2.1.3	Analyze the System and Environment	193
2.1.4	Define the Quality Characteristics	193
2.1.5	Specify the Required Documentation	194
2.1.6	Define External Quality Characteristics	194
2.2	Specify the Evaluation activity	194
2.2.1	Specify the Quality Attributes	194
2.2.2	Define the Software Component Techniques Model (SCTM)	194
2.2.3	Analysis of the Evaluation Techniques	195
2.2.4	Define Goal-Question-Metric (GQM)	195
2.2.5	Establish Punctuation Level of Metrics	195
2.3	Design the Evaluation activity	196
2.3.1	Document the Evaluation Technique/Method	196
2.3.2	Select (or Develop) Tools	196
2.3.3	Define the Environment	196
2.3.4	Develop the Evaluation Schedule	197
2.3.5	Define the Evaluation Cost	197
2.4	Execute the Evaluation activity	197
2.4.1	Collect Data	197
2.4.2	Analyze the Results	198
2.4.3	Evaluation Report	198
3.	References	199

1. Introduction

<This section should present a brief introduction of the component that will be submitted to the evaluation, the context and motivation to do so.>

1.1 Overview of the Component

<This section will present a brief overview of the component, presenting the problem that the component solves, in which domain it works, information about its internal organization and in which architecture it was developed.>

1.2 Conventions, terms and abbreviations list

This section presents the Abbreviations list used in this document.

Term	Description
Container	Environment that the component should be deployed
OS	Operational System

2. Software Component Evaluation

<This section presents the activities used to execute the software component evaluation process module which was described during this thesis. The other modules will be used during some sections of this process as required. Next section will present the steps that should be followed to evaluate the component quality.>

2.1 Establish Evaluation Requirements activity

<This activity describes all the requirements that should be considered during the component evaluation.>

2.1.1 Form an Evaluation Team

<This step presents the evaluation team that will execute the component evaluation.>

Table 1. Team Evaluation.

Evaluation Team	Stakeholder
Alexandre Alvaro	Evaluation Responsible
Valdir Alvaro	Software Development expert
Maria da Graça	Software Architecture specialist
Denise Alvaro	Application Server specialist
Fabio Henrique	Software Engineering Specialist in Java Programming Language
Eduardo Monteiro	Software Engineering Specialist in Java Programming Language

2.1.2 Define the Goals and Scope

<This step should answer some questions like: (i) What does the evaluation expects to achieve?; (ii) What are the responsibilities of each member of the team?; (iii) When should the evaluation finish?; (iv) What constraints must the evaluation team adhere to?; and (v) What is the related risk of the component to its target domain?>

This step should contain the goals of the evaluation; scope of the evaluation; component and domain risk-level; statement of commitment from both stakeholder(s) and customer(s); and summary of decisions that have already been made.>

2.1.3 Analyze the System and Environment

<This step should describe the whole environment of evaluation as precisely as possible. Additionally, the team members should answer the following questions: (i) How much effort will be spent to provide the whole infra-structure to evaluate the component? How is the complexity and what are the constraints of this environment?; (ii) What is the size of the selected systems (if available)? What is(are) the target domain(s) of those systems?; (iii) What is the impact of the software component in the selected system?; and (iv) What are the component dependencies?>

2.1.4 Define the Quality Characteristics

<This step defines the quality characteristics and sub-characteristics that will be used to evaluate the component quality. Still on, the team evaluation should define the importance level

of each characteristic defined according to this classification: *1-Not Important; 2-Indiferent; 3-Reasonable; 4-Important; 5-Very Important.*>

Example:

Table 2. Characteristics and Sub-Characteristics defined.

Characteristics	Sub-Characteristics	Importance
Functionality	Accuracy	4
Functionality	Security	3
...

2.1.5 Specify the Required Documentation

<This step describes which documents are necessary (essential) to execute the component evaluation. After that, the Customer should be contacted in order to provide those documents before the evaluation starts.>

2.1.6 Define External Quality Characteristics

<This step describes the characteristics that are not presented on the Component Quality Model (CQM), presented on Chapter 5, and should be considered to evaluate any component quality aspects. After defining the characteristics, it is interesting to complement the table 2 with the new quality characteristics and to define its relevance to the component quality evaluation.>

2.2 Specify the Evaluation activity

<This activity describes how each quality attribute will be evaluated and which techniques and metrics will be used/collected.>

2.2.1 Specify the Quality Attributes

<This step complements the Table 2 with quality attributes for each sub-characteristic as show in Table 3. The quality attributes could be selected from CQM also. In this way, the quality aspects of the component are completely developed.>

Example:

Table 3. Importance related for each characteristic.

Characteristics	Sub-Characteristics	Quality Attributes	Importance
Functionality	Accuracy	Correctness	4
Functionality	Security	Data Encryption	3
...

2.2.2 Define the Software Component Techniques Model (SCTM)

<During this step the evaluation team will define which level should be considered to evaluate the quality characteristics proposed earlier (the guidelines for selecting evaluation level could help the evaluation team in this task). Chapter 6 presented the SCTM model and the correlation between those evaluation techniques X quality attributes presented on CQM. Thus, it could be interesting to put another column on the Table 3 in order to show which techniques are interesting to evaluate the proposed quality attributes as show in Table 4.. Of course, these techniques will be based on the level that was defined by the evaluation team.>

Example:

Table 4. Evaluation techniques defined.

Characteristics	Sub-Characteristics	Quality Attributes	SCTM Level / Evaluation Technique	Importance
Functionality	Accuracy	Correctness	II. Black-Box Testing	4
Functionality	Security	Data Encryption	III. Code Inspection	3
...

2.2.3 Analysis of the Evaluation Techniques

<During this step, the evaluation team will analyze the Table 4 in order to define if the evaluation techniques proposed are useful or if it other(s) technique(s) not supported by SCTM are needed. If true, the evaluation team should establish this “new” technique, describe it, reference it from the literature and add in the Table 4.>

2.2.4 Define Goal-Question-Metric (GQM)

<This step will define all metrics necessary in order to collect the data and analyze it at the end of the evaluation process. It should be defined, at least: one metric for each quality attribute proposed on the Table 4; and one metric for each module of the framework.>

Example:

Table 5. GQM example for Correctness Quality Attribute.

Functionality	
Sub-Characteristic	Accuracy
Quality Attribute	Correctness
Goal	Evaluates the percentage of the results that were obtained with precision
Question	Based on the amount of tests executed, how many test results returned with precision?
Metric	Precision on results / Amount of tests
Interpretation	0 <= x <= 1; closer to 1 being better

2.2.5 Establish Punctuation Level of Metrics

<Based on the last step, this one will define the score level of each metrics defined earlier, based on the interpretations defined for each metric.>

Example:

Table 6. Example of Score Level in the GQM definition.

Functionality	
Sub-Characteristic	Accuracy
Quality Attribute	Correctness
Goal	Evaluates the percentage of the results that were obtained with precision
Question	Based on the amount of tests executed, how many test results returned with precision?

Metric	Precision on results / Amount of tests
Interpretation	$0 \leq x \leq 1$; which closer to 1 is better
Score Level	<ul style="list-style-type: none"> • 0 – 0.3: Not acceptable • 0.31 – 0.6: Reasonable quality • 0.61 – 1: Acceptable

2.3 Design the Evaluation activity

<This activity describes the whole configuration of the environment that will be used to evaluate the component quality.>

2.3.1 Document the Evaluation Technique/Method

<During this step the evaluation team will document the techniques used during the component evaluation (those described on section 2.2.2.). The whole team must have knowledge in each specific technique proposed early in order to help during the documentation and for help the members that don't know so much about certain technique.>

2.3.2 Select (or Develop) Tools

<This step will define how the evaluation team will execute the evaluation techniques defined earlier. They can use a tool, or develop a tool, or a specific method, or a framework, etc. in order to evaluate the quality attributes. After defining the tools/method/technique/etc, it is interesting to document it so that the whole team can better understand and use it.

The team could use the Table 4 in order to document which tool/method/process/technique will evaluate each technique, as shows in Table 7:>

Example:

Table 7. Definition of the Tools that should be used during evaluation.

Characteristics	Sub-Characteristics	Quality Attributes	SCTM Level / Evaluation Technique	Importance	Tool used
Functionality	Accuracy	Correctness	II. Black-Box Testing	4	Junit ²⁰ , FindBugs ²¹
Functionality	Security	Data Encryption	III. Code Inspection	3	PMD ²²
...

2.3.3 Define the Environment

<This step describes, as precisely as possible, the whole environment to evaluate the component. There are two options: (i) specify the system that the component will work in order to evaluate the quality of the components executed in these systems provided by the costumer; (ii) specify a well-defined environment for the component to be executed and analyzed.>

²⁰ <http://www.junit.org>

²¹ <http://findbugs.sourceforge.net>

²² <http://pmd.sourceforge.net>

2.3.4 Develop the Evaluation Schedule

<This step will provide the time spent to evaluate the component quality and the activities to be executed for each stakeholder defined on section 2.1.1., as shows in Table 8.>

Example:

Table 8. Component Evaluation Scheduler.

Activities	01/11/2007	05/12/2007
Configure the environment	Alexandre Alvaro	
Develop the black-box test case	Valdir Alvaro, Maria da Graça	
Define the static analysis that will be considered	Denise Alvaro	
Analyze the source-code	Fabio Henrique, Eduardo Monteiro	
Measure the whole process using the metrics defined	Alexandre Alvaro	
Generate the final report		Alexandre Alvaro

2.3.5 Define the Evaluation Cost

<Based on the team expertise, this step describes the evaluation costs, based on: the number/cost of each stakeholder and the time spent by the stakeholder on each activity of the evaluation. If the team evaluation has more expertise in previous evaluations or in other kinds of costs estimation, they should use it to develop the evaluation cost. >

2.4 Execute the Evaluation activity

<This activity will execute the whole planning of the component evaluation. First the team evaluation will configure the environment, after that it will execute the evaluation in order to Analyze if the component has the desired quality level or not.>

2.4.1 Collect Data

<During execution of the evaluation (last section), all data provided is collected using the metrics defined in section 2.2.4. A table should be used to store those values in order to be further analyzed. An example is show in Tale 9.>

Example:

Table 9. Table to document the results obtained during component evaluation.

Characteristics	Sub-Characteristics	Quality Attributes	SCTM Level / Evaluation Technique	Importance	Tool used	Results
Functionality	Accuracy	Correctness	II. Black-Box Testing	4	Junit, FindBugs	0.7
Functionality	Security	Data Encryption	III. Code Inspection	3	PMD	0.8
...

2.4.2 Analyze the Results

<During this step the evaluation team will analyze all data collected in order to provide the quality level of the component. Some adjustments could be done in this step because there are some quality attributes that could influence, in a positive or negative way, other quality attributes. Thus, these questions should be carefully analyzed and considered by the evaluation team.

Moreover, the evaluation team should consider the importance level of each quality attribute in a way that different weights could be applied for each results obtained.>

2.4.3 Evaluation Report

<In this step, the evaluator responsible for the component evaluation will develop a report that contains the information obtained during the previous steps and the evaluation team should provide some comments in order to the customer improve their component.

The evaluator should consider if the component achieves the required quality to be considered in the level in which it was evaluated. This could be achieved through the analysis of the score level of each metric defined during the component evaluation process execution.>

3. References

<This section will provide the references to tools, processes, techniques, methods cited during this documents, in such format :>

[1] Authors, Title; Conference/Journal (if applicable); Date;

Appendix C.

Questionnaires used in the Experimental Study

This appendix presents the two questionnaires used in the experimental study.

QT1 – INDIVIDUAL QUESTIONNAIRE FOR THE PARTICIPANTS OF THE EXPERIMENT

Date: __/__/____

Course: () Computer Science () Computer Engineering () Informatics
Bachelor () Data Processing

() Other: _____

Degree: () Graduate () M.Sc. () PhD. () Specialization

In which of the categories below do you belong, in relation to software quality?

() I have no experience in software quality.

() I have developed some projects during graduation/postgraduation courses, using software quality techniques.

() I have developed, professionally, some projects using any kind of software quality techniques (up to 3).

Appendix C – Questionnaires used in the Experimental Study

() I have developed, professionally, several projects using any kind of software quality technique (more than 3).

() I'm coordinator/manager of software quality in my company (up to 2 years)

() I'm coordinator/manager of software quality in my company (more than 2 years)

() Other, specify: _____

Please, inform which courses you attended in the software quality / software evaluation / software engineering / software reuse areas

Do you know and/or work with any software quality techniques? Which one(s)?

How much time do you have in experience in each technique?

Check you experience or the activities (positions) that you exercise (or have exercised), in the software development area:

- ☐ Systems analyst
- ☐ Software architect
- ☐ Software engineer
- ☐ Components developer
- ☐ Applications developer (with components)
- ☐ Tests engineer
- ☐ Quality engineer
- ☐ Configuration engineer
- ☐ Project manager
- ☐ Teacher (university) in informatics (reuse-oriented disciplines)
- ☐ Others: _____

In how many developments of applications using some software quality technique have you participated?

Large complexity:

- ☐ None ☐ 1 - 2 ☐ 3 - 7 ☐ More than 7

Medium complexity:

- ☐ None ☐ 1 - 2 ☐ 3 - 7 ☐ More than 7

Small complexity:

- ☐ None ☐ 1 - 2 ☐ 3 - 7 ☐ More than 7

Informing the amount of training on software quality you have, by checking the correspondent items and quantities below (excluding the course ministered in this semester)

Courses (up to 8 hs):

☐ None ☐ 1 - 2 ☐ 3 - 7 ☐ >7

Courses (up to 40 hs):

☐ None ☐ 1 - 2 ☐ 3 - 7 ☐ >7

Courses (more than 40 hs):

☐ None ☐ 1 - 2 ☐ 3 - 7 ☐ >7

Symposiums/Conferences:

☐ None ☐ 1 - 2 ☐ 3 - 7 ☐ >7

Publications of national papers:

☐ None ☐ 1 - 2 ☐ 3 - 7 ☐ >7

Publications of international papers:

☐ None ☐ 1 - 2 ☐ 3 - 7 ☐ >7

Others: _____

In which area are you most interested:

☐ Software Engineering ☐ Networking/Distributed Systems ☐ Databases

☐ Artificial Intelligence ☐ Hypermedia ☐ Computers architecture

☐ Graphical Computing

☐ Other: _____

Observations or comments: (please use the back page if the space below is insufficient)

QT2 – INDIVIDUAL QUESTIONNAIRES FOR THE PARTICIPANT OF THE EXPERIMENT

Regarding the component evaluation process, answer:

1) Which difficulties did you find in the *Establish Evaluation Requirements*? (justify)

2) Which improvements would you suggest for the *Establish Evaluation Requirements* activity?

3) Which difficulties did you find in the *Specify the Evaluation*? (justify)

4) Which improvements would you suggest for the *Specify the Evaluation* activity?

5) Which difficulties did you find in the *Design the Evaluation*? (justify)

6) Which improvements would you suggest for the *Design the Evaluation* activity?

7) Which difficulties did you find in the *Execute the Evaluation*? (justify)

8) Which improvements would you suggest for the *Execute the Evaluation* activity?

9) Do you have any other consideration about the whole process?

Regarding the Component Quality Model (CQM), answer:

10) Do you consider that the Component Quality Characteristics presented on the Component Quality Model (CQM) are sufficient to measure the component quality? If you needed to propose any other quality characteristic during the evaluation process that is not covered in the model, justify your decision.

Regarding the Software Component Techniques Model (SCTM), answer:

11) Do you consider that techniques provided on the Software Component Techniques Model (SCTM) are sufficient to evaluate the component quality? If you needed to propose any other quality technique that is not covered in the model, justify you decision.

12) Do you think that *Guidelines for selecting evaluation level* helped you during the definition of the SCTM level? Why?

13) Which improvements would you suggest for the *Guidelines for selecting evaluation level*?

Regarding the Metrics Framework, answer:

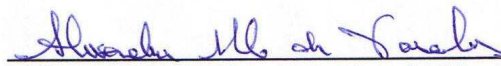
14) Do you think that the Metrics Framework (using the Goal-Question-Metrics (GQM) paradigm) and which contains a set of metrics examples to help the evaluation team was fundamental during the measurement proposals? (justify)

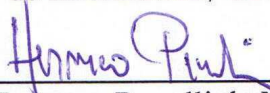
Regarding the whole process, answer:

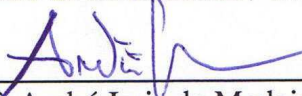
15) After using the whole Software Component Quality Framework, do you believe that the component quality could be precisely evaluated/measured through this Framework?

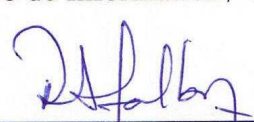
16) Other suggestions, difficulties, comments, etc. about the Framework.


Tese de Doutorado apresentada por **Alexandre Álvaro** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**A Software Component Quality Framework**", orientada pelo **Prof. Silvio Romero de Lemos Meira** e aprovada pela Banca Examinadora formada pelos professores:


Prof. Alexandre Marcos Lins de Vasconcelos
Centro de Informática / UFPE

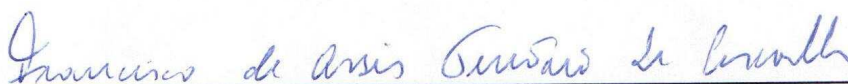

Prof. Hermano Perrelli de Moura
Centro de Informática / UFPE


Prof. André Luis de Medeiros Santos
Centro de Informática / UFPE


Prof. Ricardo de Almeida Falbo
Departamento de Informática / UFES


Profa. Renata Pontin de Mattos Fortes
Deptº de Ciência da Computação e Estatística / USP

Visto e permitida a impressão.
Recife, 4 de abril de 2009.


Prof. FRANCISCO DE ASSIS TENÓRIO DE CARVALHO
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.