
Pós-Graduação em Ciência da Computação

# “Software Component Certification: A Component Quality Model”

By

*Alexandre Alvaro*

MSC DISSERTATION

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

RECIFE, MES/ANO

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ALEXANDRE ALVARO

# “Software Component Certification: A Component Quality Model"

*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 MESTRE EM CIÊNCIA DA COMPUTAÇÃO.*

*A MASTER DISSERTATION PRESENTED TO THE FEDERAL UNIVERSITY OF PERNAMBUCO IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MSC. IN COMPUTER SCIENCE.*

ADVISOR: Silvio Lemos Meira

RECIFE, OCTOBER/2005

Alvaro, Alexandre
Software component certification : a component quality model / Alexandre Alvaro . – Recife : O Autor, 2005.
124 folhas. : il., fig., tab.

Dissertação (mestrado) – Universidade Federal de Pernambuco. CIn. Ciência da Computação, 2005.

Inclui bibliografia.

1.Engenharia de software – Reuso de software. 2. Componentes – Qualidade e certificação. 3. Modelo de qualidade de componentes – Proposta de utilização . I. Título.

| | | |
|---|---|---|
| 004.052.42 | CDU (2.ed.) | UFPE |
| 005.1 | CDD (22.ed.) | BC2005-646 |

*To my wonderful and well-beloved family:*
*Valdir and Maria*
*Denise, Fabio and Eduardo.*

# cknowledgements

This dissertation 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 Lemos Meira, who motivated, encouraged and trusted me when I came to Recife through his financial support. He was always ready with good and precise feedback. I am very thankful for all the “doors” that he opened for me. He is the advisor that all post-graduate students would like to work with.

In particular, I would like to thank my committee members, Alexandre Vasconcelos and Sergio Soares, for the time input and many stimulating discussions. Their comments greatly helped to improve my dissertation.

I am thankful, again, to Alexandre Vasconcelos, for giving me insights in software quality, which helped in the definition of the component quality model, and for always being available to read my papers.

I am deeply grateful for all of my friends that shared some moments with me. Thanks Eduardo Almeida, which is the person that believed in me more than myself. He is the person that encouraged me to came to Recife, to create the Software Reuse group (RiSE) at UFPE. Thanks for some discussions about software reuse, software components, component certification, repository systems, software architecture, some papers and, the most important, the parties. Thanks Daniel Lucrédio for all of your attention during the whole 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

dissertation. Thanks for Vinicius Garcia, for your happiness, for discussions about the courses at UFPE and for his company in some moments. Thanks for Ricardo Ramos, for all of your company in some moments in Recife, some moments that we go out – sometimes we go out just to drink and talk about the universe –, the discussions of some points about our lives and for all the traveling that we made, knowing very beautiful and historic places. His company here made my life happier. Thanks for Cynthia Roberta, who is always available for helping me in some bad or good moments. Thanks for my friends at Sorocaba-SP (Bruno, Neto, Paulo, Marcos and Alisson). Even with the distance, they were always available for talking and giving me force for continuing this work. Still on, thanks for my friends Andre, Cleber and Hamilton, who are always interested in my future ways, in order to us 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 (Vanilson Burégio, Jorge Mascena, Taciana Amorim, Ana Paula Cavalcanti, Juliana Dantas, Eduardo Cruz, Frederico Durão and Ednaldo Dilorenzo) for their support in this whole time. Particularly Jorge Mascena, for making available the C.E.S.A.R. software components to me, and Juliana Dantas, for giving me support at C.E.S.A.R.

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 means to put my work in a real practice environment. And to the Informatics Center at UFPE for making available its infra-structure.

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 need them the most, even so distant. Sorry for staying so far during this long time, the homesickness was enormous and don’t worry, one day I will come back to our hometown. I love you two. Grateful thanks to my lovely sister, Denise, who is always present (virtually) 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. Very thanks Deba, I love you. Very special thanks to Fabio, for his company in some moments in Sorocaba-SP, for all the barbecues, beers, joking and talking during the little time that I was there. Thanks Fabão. And thanks for the more new people of my family, Eduardo, the most beautiful and likeable baby that I know. They are the most important people in all of my life. I hope 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 and Eduardo baby. Without them, I would not be where I am now. They will always be in my heart!!

*To be great, be whole; exclude*

*Nothing, exaggerate nothing that is you.*

*Be whole in everything. Put all you are*

*Into the smallest thing you do.*

*The whole moon gleams in every pool,*

*It rides so high.*

Fernando Pessoa
*Ricardo Reis, Odes*

# esumo

Desenvolvimento de Software Baseado em Componentes tem sido amplamente adotado na academia e na prática, representando assim um mercado promissor para a indústria de software. A perspectiva de redução do custo e do ciclo de desenvolvimento do software tem sido a principal motivação para esta expansão. Entretanto, inúmeros problemas técnicos ainda permanecem sem solução antes mesmo que a indústria de componentes de software alcance a maturidade de outras indústrias de software. Problemas como a seleção de componentes, a carência de catálogos de componentes formalizados e a falta de informações sobre a qualidade dos componentes desenvolvidos trazem novos desafios para a comunidade de engenharia de software. Por outro lado, a área de certificação de componentes de software é relativamente imatura e necessita de consideráveis pesquisas para o estabelecimento de um padrão para certificação de componentes de software. Assim, esta dissertação apresenta um modelo de qualidade de componentes, baseada em características consistentes e bem definida, atributos de qualidade e métricas relacionadas para avaliação dos componentes. Um estudo experimental foi desenvolvimento visando analisar a viabilidade de utilização do modelo. Os resultados obtidos mostrar-se-iam que o modelo é um interessante candidato para avaliação da qualidade em componentes de software, e trabalhos futuros estão sendo planejamentos para que a evolução do modelo alcance o nível de maturação necessário e torne-o diretamente aplicável à indústria de software.

# bstract

Component-based software development is becoming more generalized, representing a considerable market for the software industry. The perspective of reduced development costs and shorter life cycles acts as a motivation for this expansion. However, several technical issues remain unsolved before the software components industry reaches the maturity as other software industries. Problems such as component selection, the lack of component catalogs formalization and the uncertain quality of third-party developed components bring new challenges to the software engineering community. On the other hand, software component certification is still immature and much research is needed in order to create well-defined standards for certification. This dissertation introduces a component quality model, based upon consistent and well-defined characteristics, quality attributes and related metrics for the component evaluation. An experimental study was used in order to analyze the viability of the use such of a model. The results showed that the model is a good candidate for evaluating software component quality, and future work already scheduled to continue evolving until it reaches a maturation level that makes it applicable in industry.

# ist of Figures

# ist of Tables

# cronyms

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

# Contents

# Introduction

## 1.1 Motivation

One of the most compelling reasons for adopting component-based approaches in software development, with or without objects, 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 the 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 certification. 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 certification and 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 certification should be carried out? **(ii)** what are the requirements for a certification 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 certification (Voas et al., 2000), (Morris et al., 2001).

In this context, the main goal of this dissertation is investigating effective ways to demonstrate that component certification is not only possible and practically viable, but also directly applicable in the software industry. Through certification, 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. This fact could be seen through a nationwide project launched by the

Brazilian government[1]. This project's main concerns are: to develop a robust framework for software reuse (Almeida et al., 2004a), in order to establish a standard to the component development; to develop a repository system; and to develop a component certification process. This project has been developed in a collaboration between the industry and academia (the RiSE group[2] and two other 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) that is being developed has two layers. The first layer (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 layer constitutes a fundamental step before of the introduction of the framework in the organizations. The second layer (on the right) is composed of important technical aspects related to software reuse, such as processes, environments, tools, and a certification process, which is the focus of this dissertation.

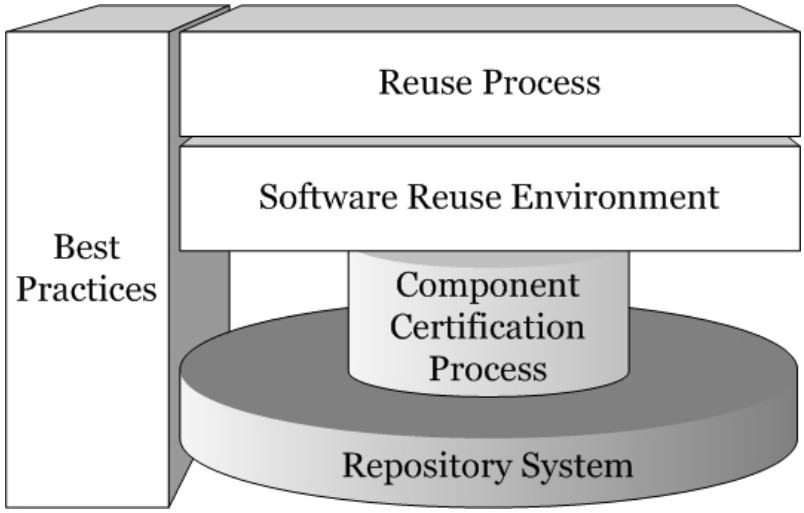


Figure 1. The Framework for Software Reuse.

However, the process of certifying components 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.,

[1] http://www.finep.gov.br//fundos_setoriais/acao_transversal/resultados/resultado_Acao_transversal_Biblioteca_de_Componentes_05_2004.PDF

[2] RiSE – Reuse in Software Engineering group – http://www.cin.ufpe.br/~rise

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 reuse. With a 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 **certification process**. Finally, a set of **metrics** are needed, in order to track the components properties and the enactment of the certification process.

These four main issues: **(i)** a Component Quality Model, **(ii)** a Certification Techniques Framework, **(iii)** a Metrics Framework, and **(iv)** a Certification Process, are the modules of a Software Component Certification Framework (Figure 2) that is being investigated as a part of the RiSE project.

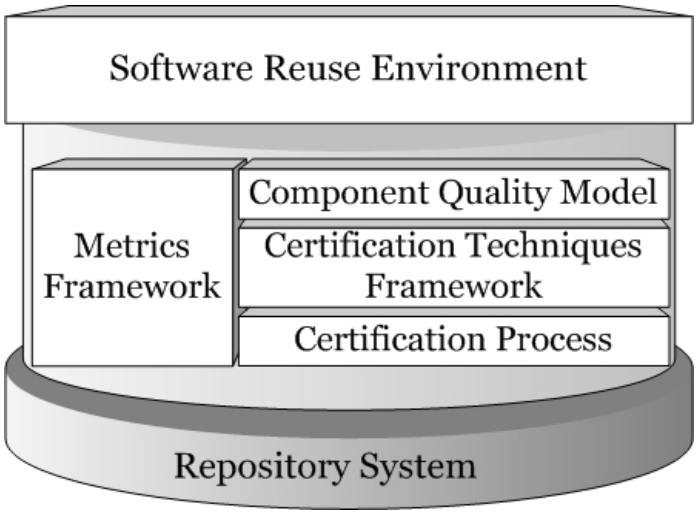


Figure 2. Software Component Certification Framework.

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.

However, the whole framework could not be fully developed in a single M.Sc. project without the risk of producing poor theoretical and practical results. Therefore, this dissertation focuses on the Component Quality Model, which is the first module that is required by the framework. The other modules are left for future works (Chapter 7).

### 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 of earlier adopters of software component technology. A summary from a Web survey of component adopters is presented in Figure 3.

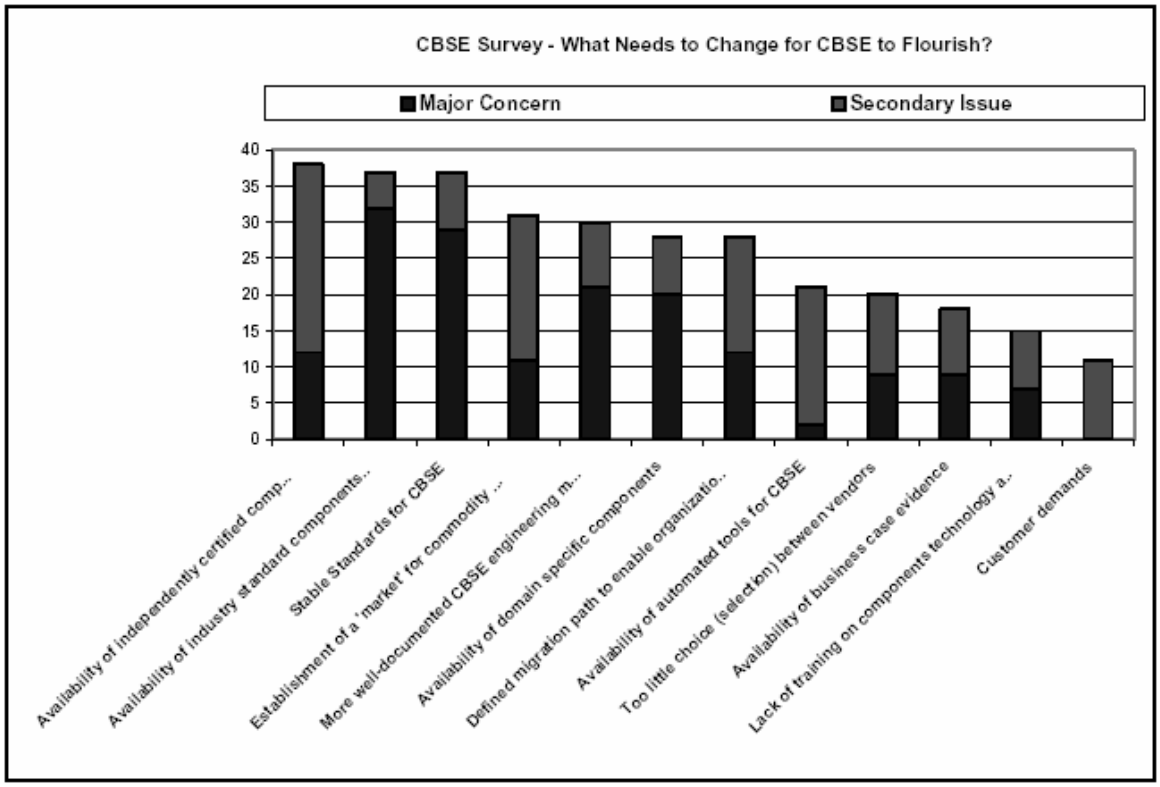


Figure 3. Summary of Survey Responses (Bass et al., 2000).

From this data and from the interviews, the study concludes 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 markets, such as *Flashline*[3] and *ComponentSource*[4], which contain, respectively, more than 13,000 and 9,000 components (four years ago they had less than 1,000 and 100 components (Trass et al., 2000), respectively), 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 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 certification reflects a natural progression of concerns: first demonstrate 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.

Still on, 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 certification 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.

[3] http://www.flashline.com
[4] http://www.componentsource.com

### 1.1.2 The Future of Software Components

Important researches on software components, such as Heineman (Heineman et al., 2000), Councill in (Heineman et al., 2001), Crnkovic (Crnkovic, 2001) and Wallnau (Wallnau, 2003) points to the future of software components is certification. These authors state that certification 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. Certified components 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 certified components 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, certification reduces the risk of system failure by providing information about a software component's risk mitigation procedures, 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 certified, the confidence that the information accompanying these components meets their requirements will be greater.

## 1.2 Problem Statement

In general, the main idea of certification is to bring quality to a certain software product, in this case software components. One of the core goals to achieve quality in component is to acquire reliability on it and, in this way, to increase the component market adoption. Normally, 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, in order to measure the quality of a

software component, it is necessary to develop a quality model, in this case, a Component Quality Model.

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 address the software products' quality issues (in particular, those from ISO and IEEE) 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 (Schmidt, 2003). Still, there are several difficulties in the development of component quality model, such as **(i)** which quality characteristics should be considered, **(ii)** how we can evaluate them and **(iii)** who should be responsible for such evaluation (Goulão et al., 2002a). In this way, there is still no well-defined standard and component quality model to perform component certification (Morris et al., 2001) (Voas et al., 2000). This fact may be due also to the relatively novelty of this area (Goulão et al., 2002a).

Although recent, some component quality models can be found into literature (more details about the models can be seeing in related works section on Chapter 7). The promising works are based on ISO 9126 (ISO/IEC 9126, 2001). This standard is a generic software quality model and it can be applied to any software product by tailoring it to a specific purpose. The main drawback of the existing international standards is that they provide very general quality models and guidelines, and are very difficult to apply to specific domains such as COTS (Commercial Off-The-Self) components and Component-Based Software Development (CBSD).

Even so, there are works in the literature that try to analyze the ISO 9126 and to propose a model that is specific for software components (Goulão et al., 2002b), (Bertoa et al., 2002), (Simão et al., 2003). The researchers aim to verify which characteristics of ISO 9126 are adequate to the component context or if new characteristics need to be added or removed from the model.

However, these models were not evaluated into academic or industrial scenarios. In fact, they are based on the researchers' experience, and the real efficiency of evaluating software components using these models remains unknown. Additionally, two works (Goulão et al., 2002b), (Simão et al., 2003) did not specify the metrics that should be used to measure the quality characteristics proposed in the model, making it difficult to use this model in any kinds of scenarios.

## 1.3 Overview of the Proposed Solution

The component market, which is a priori condition to maximize the intra-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 certification programs would help to acquire trust in the market components (Heineman et al., 2001).

Motivated by these ideas, the main goals of this dissertation were to define a Component Quality Model, to identify its characteristics, the sub-characteristics, the quality attributes and the related metrics that compose the model. After that, an evaluation of the model was accomplished, in order to analyze the gap that exists between the component quality information required to certify software components and the component quality information provided by the main component markets and a Brazilian software factory.

Also, the software components that are being developed in the project cited earlier (Section 1.1) will be evaluated using this component quality model proposed. This will help the future evolution of the model, constantly testing and validating its attributes with the real components that are developed by the software factories that are engaged to the project.

## 1.4 Statement of the Contributions

Basically, the main contributions of this research are:

- A survey of the state-of-the-art of component certification (Alvaro et al., 2005a), in an attempt to analyze this area and possible trends to follow;
- Definition of the key CBD characteristics that would compose the component quality model;
- A development of a well-defined Component Quality Model (CQM) to assure quality to software components (Alvaro et al., 2005b), (Alvaro et al., 2005c), (Alvaro et al., 2005d); and
- A preliminary evaluation of the CQM (Alvaro et al., 2006a).

## 1.5 Out of Scope

In order to assure quality to whatever kind of software product it is necessary to use a certain quality model. Since there is still no well-defined component quality model to perform component certification and this is the first achievement of this area, this dissertation is focused on this model.

However, due to scope limitations, there are other possibilities and work directions that were discarded in this dissertation. For instance, 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 dissertation 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.

Other work, proposed by the 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 CBSD in order to determine which quality properties the customer can

expect from the system that will be developed using certain components. This dissertation does not consider this approach for the same reasons it does 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, this is a recent approach, still under development. One of the SEI's main researchers, Kurt Wallnau (Wallnau, 2005), even states that the viability of this approach is still unknown. Due to this immaturity, this dissertation focuses on other aspects of the component certification process.

And finally, as discussed earlier, although a certification process requires all four modules of the Component Certification Framework presented in Section 1.1, this dissertation covers only one module: the Component Quality Model. The other modules of the framework will be considered in future works, as described in Chapter 7.

## 1.6 Organization of the Dissertation

In order to find some insights for the questions that stand out, this dissertation investigates the current state-of-the-art on software component certification, in an attempt to analyze this area and with the objective of looking for research directions to follow. Besides this chapter, this dissertation 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 certification 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 certification area that was performed. The failure cases that can be found in the literature are also described in this chapter.

Chapter 5 describes the proposed Component Quality Model, showing its characteristics, its sub-characteristics, the quality attributes and the metrics that are related to the model.

Chapter 6 presents the formal case study of how much of the information required assessing components is actually available in the most popular COTS components markets and into a component repository of a Brazilian Software Factory.

Chapter 7 summarizes the main contributions of this work, presents the related works, the concluding remarks and directions for future work.

# 2 Software Reuse

Reuse products, processes and other knowledge will be the key to enable the software industry to achieve the dramatic improvement in productivity and quality that is required to satisfy anticipated growing demands (Basili et al., 1991). However, to fail 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 et al., 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 down.

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

According to Basili er al. (Basili et al., 1991), the following assumptions should be considered in 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 or 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; and
- **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 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; and
- **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 has 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 et al., 1994), (Joos, 1994), (Griss et al., 1995), (Morisio et al., 2002). An interesting survey that relates advantages, success and failure cases, myths and inhibitors for software reuse adoption is presented in (Almeida et al., 2005a).

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, 2005), Component Object Model (COM) (Microsoft COM, 2005), 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 the promises of some organizations to promote reuse and the component markets have grown exponentially due to demand. However, the component area is still immature and future research is needed.

### 2.1.1 Software Components

The exactly concept of component in CBD is not yet a consensus, due to the relative novelty of this area[1]. 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 the 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.

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.

[1] 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).

However, the concept presented in (Szyperski, 2002) is sufficient to establish a satisfactory definition about what is a component in CBD, and will be used as basis in this dissertation:

"*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 client (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 presented. Additionally, this Chapter presents definitions on software components, and a brief explanation of software components assumptions.

# 3 Software Quality and Certification

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). 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, 2004):

- **Software Product Quality:** aiming to assure the quality of the generated product (e.g. ISO/IEC 9126 (ISO/IEC 9126, 2001), ISO/IEC 12119 (ISO/IEC 12119, 1994), ISO/IEC 14598 (ISO/IEC 14598, 1998), (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 (CMM) (Paulk et al., 1993), Capability Maturity Model

Integrated (CMMI) (CMMI, 2000), Software Process Improvement and Capability dEtermination (SPICE) (Drouin, 1995), among others).

Focusing on software product quality, according to the standard ISO 8402 (ISO/IEC 8402, 1994), which is a part of the ISO 9000 standard (ISO/IEC 9000, 1994), 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 complementing, mainly to better define the terms *entity* and *explicit and implicit necessities*. *Entity* is a product whose quality needs to be assured; the *explicit* necessities are the conditions and objectives proposed 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 dissertation 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 1 shows the main national and international standards in this area.

Table 1. Main software quality standards.

| Standards | Overview |
|---|---|
| ISO 9126 | Software Products Quality Characteristics |
| ISO 14598 | Guides to evaluate software product, based on practical usage of the ISO 9156 standard |
| ISO 12119 | Quality Requirements and Testing for Software Packages |
| 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 9000-3 | Quality Management and Assurance. Application of the ISO 9000 standard to the software development process (evolution of the NBR ISO 8402). |
| CMMI (Capability Maturity Model Integration) | 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 15504 | It is a framework for the assessment of software processes. |

The software market has grown in the last eleven years, as well as the necessity of producing software with quality. Thus, obtaining quality certificates has been a major concern for software companies. Figure 4 (Weber et al., 2002) shows how this tendency influenced the Brazilian software companies from 1995 until 2002.

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 CMM, 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 a very large different of software companies around the world.

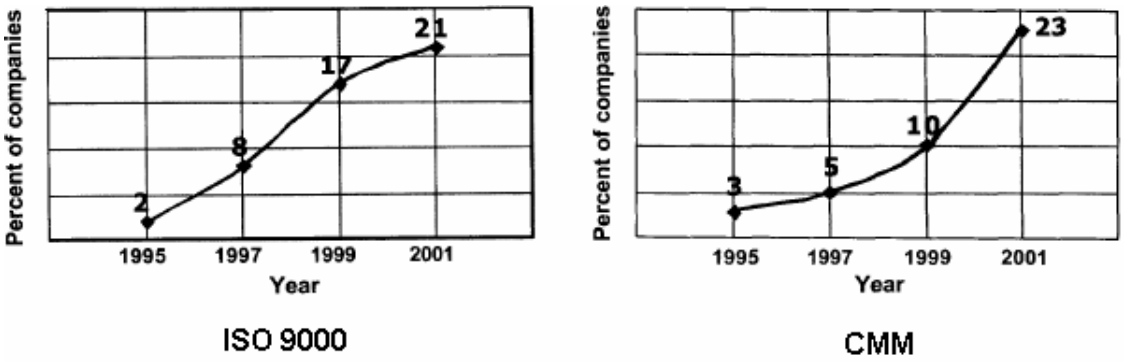


Figure 4. Experiments with the ISO 9000 and CMM in the Brazilian software companies (Weber et al., 2002)

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 component quality assurance may be seen in the ISO 9126, described next.

## 3.1 ISO/IEC 9126

The International Organization for Standardization (ISO) and the International Electro-technical Commission (IEC) have developed standards and metrics for software product evaluation. ISO/IEC developed ISO/IEC 9126, which is composed of: ISO/IEC 9126-1 (quality model) (ISO/IEC 9126, 2001), 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).

The objective of this series of standards is to provide a framework for the evaluation of software quality. ISO/IEC 9126 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. ISO/IEC 9126 defines a quality model that comprises six characteristics and 27 sub-characteristics (Table 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 2. Characteristics and Sub-Characteristics in ISO/IEC 9126.

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

The internal and external quality characteristics are modeled with the same set of six characteristics: functionality, reliability, usability, efficiency, maintainability and portability. The usage quality characteristics (characteristics that can be obtained from the developer or designer feedback of the product) are modeled with four characteristics: effectiveness, productivity, security and satisfaction.

However, the main drawback of the existing international standards, in this case the ISO/IEC 9126, is that they provide very generic quality models and

guidelines, which are very difficult to apply to specific domains such as COTS components and CBSD. Thus, the quality characteristics of this model should be analyzed in order to define and adequate to the component quality characteristics context.

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.

However, in CBSE, there is a third kind of stakeholder: the component consumer, which may not have access to the code (black-box reuse). Even with a good component quality model, the consumer is often unable to assure the quality attributes himself. Thus, he must trust that the component has the desired quality levels. Next section is dedicated to discuss the main concepts involving software components certification, which is an attempt to achieve this kind of trust.

## 3.2 Software Component Certification

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 dissertation:

*“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 with basis 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 with concern for 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 CBSD, 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 et al., 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 5.

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. **The definition of such a model is the main objective of this dissertation.**

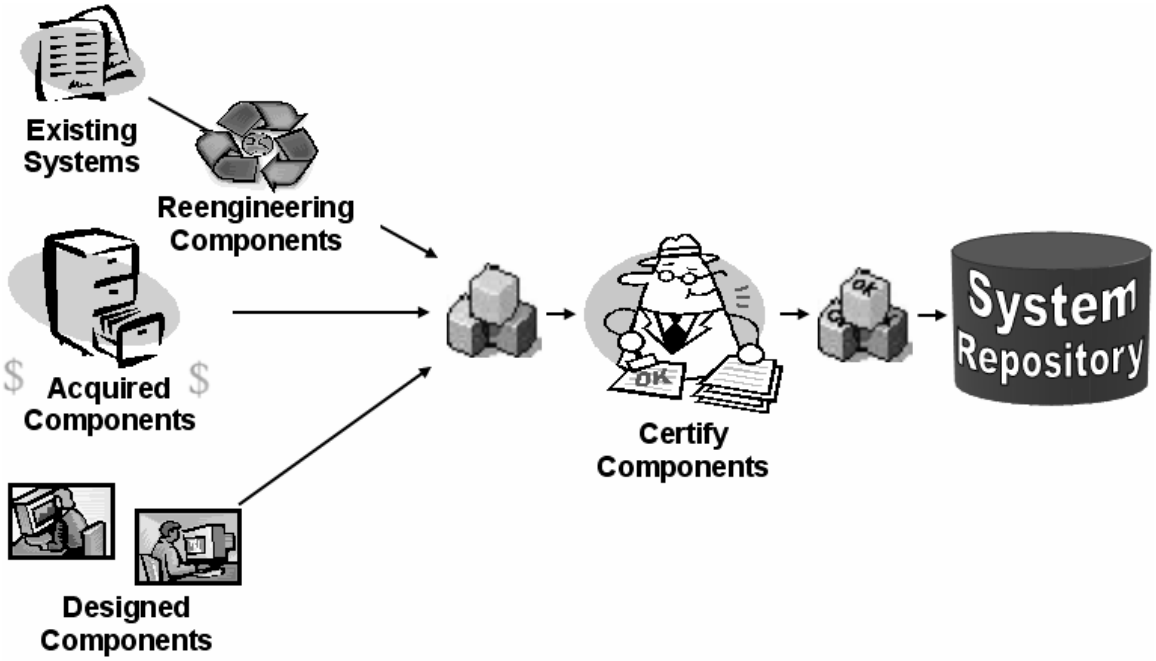


Figure 5. Process of obtaining, certifying and storing components

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).

## 3.3 Summary

This Chapter presented the main concepts about software quality and, in the context of this dissertation, quality related to software components. It also presented ISO/IEC 9126, a software quality model 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 to insights 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 et al., 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.

## 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)[1], 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 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 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.

[1] http://www.trusted-components.org

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 above.

Another work involving component test may be seen in (Wohlin et al., 1998), where Wohlin et al. 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 et al. 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) Verificator**, 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 and 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 maturity 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 trusty 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 Heineman et al. (Heineman 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 <*property*,*value*,*credibility*>. 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 contain four activities, as follows: **(i) Planning the evaluation**, where the evaluation team is defined, the stakeholders are identified, the required resources is estimated and

the basic characteristics of the evaluation activity is determined; **(ii) Establishing the criteria**, where the evaluation requirements are identified and the evaluation criteria is constructed; **(iii) Collecting the data**, where the component data are collected, the evaluations 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.

However, the proposed process is an ongoing work and, until now, no real case study was accomplished in order to evaluate this process, becoming unknown the real efficiency to evaluate software components.

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 is 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, 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 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 6 shows an overview of this model.

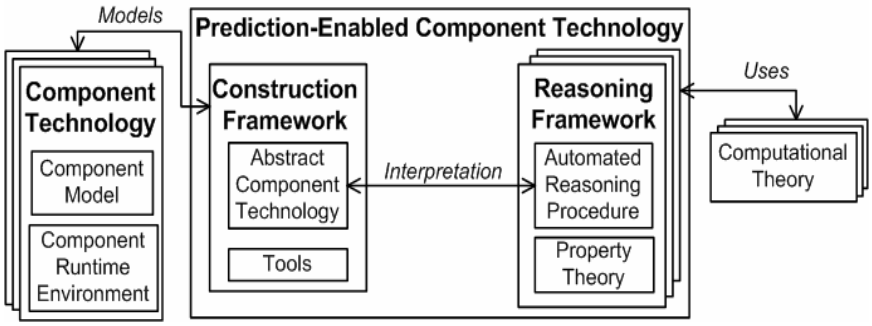


Figure 6. Structure of Prediction-Enabled Component Technology (Wallnau, 2003).

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 an ongoing work in the current SEI research framework. 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 other remarks. 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 in 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.

However, these two directions are still ongoing research. The effort to develop a component certification standard is only in its beginning.

## 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 certificating 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 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."[2], as the basis for the Common Criteria[3], aimed at certifying security features of components. Their effort was not crowned with success, at least partially because it had defined no

[2] http://www.radium.ncsc.mil/tpep/library/tcsec/index.html
[3] http://csrc.nist.gov/cc

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 Summary of the Study

Figure 7 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 2004 (Figure 7). 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, its requirements and discussion about how to achieve component certification (represented by a square). The arrows indicate that a work was extended by another.

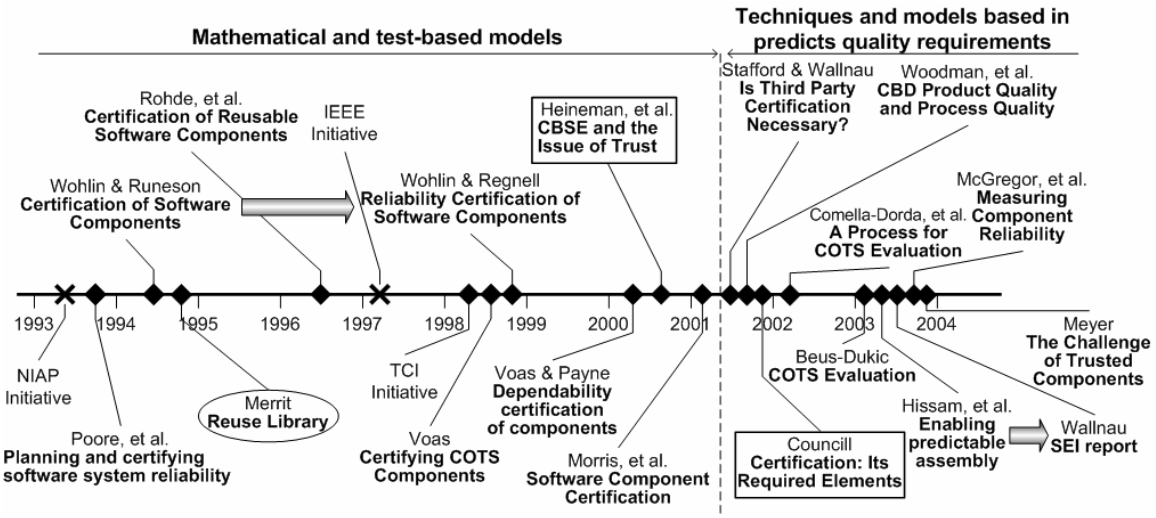


Figure 7. Research on software component certification timeline.

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

# 5 Software Component Quality Model

Component-Based Software Development (CBSD) is being used in a wide variety of application areas and the correct operation of the components is 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. A risk of selecting a product with unknown quality properties is no longer acceptable and, when happened, may cause catastrophic results (Jezequel et al., 1997). Thus, the software components quality evaluation has become an increasingly essential activity in order to bring reliability in (re)using software components.

Unfortunately, the dream of formal verification is unrealistic and achievable at best only in the case of small source code components. The software factories cannot expect to formally guarantee the correctness of software components markets in order to build large scale software systems. Thus, software factories have to build its software systems with components that may be faulty, i.e. component with unknown quality (Sametinger, 1997). On the other hand, according to Stafford et al. (Stafford et al., 2001), commercial component vendors are not inclined to formally specify their software components too, and it is not certain that it would be cost effective for them to do so.

However, it is strictly necessary to evaluate software component quality in order to support the component markets evolution and maturity. And most of the research dedicated to software components is focused on their functional aspects (i.e. component specification, component development, component tests, etc.). In this dissertation, the main concern is the evaluation of software

component quality, within the framework of a Component Quality Model (CQM). However, there are several difficulties in the development of such a model, such as: **(1)** which quality characteristics should be considered, **(2)** how would be evaluating them and **(3)** who should be responsible for such evaluation (Goulão et al., 2002a).

In general, the main idea behind certification is to bring quality to a certain software product, in this case software components. 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. Normally, 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. Thus, this Chapter presents a Component Quality Model, its characteristics and sub-characteristics, the quality attributes and the related metrics that compose the model.

## 5.1 The Component Quality Model

The CQM proposed in this dissertation is based on ISO 9126 (ISO/IEC 9126, 2001) 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 9126, a CQM was developed (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;
- **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, complementing the quality characteristics of this model.

Although the model is proposed following the ISO 9126 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 characteristic proved to be not interesting to evaluate components was eliminated;
- The name of one of the characteristics was changed in order to adequate it to the component context;
- Another level of characteristics was added, containing relevant marketing information for a software component certification process; and
- Some characteristics that complement the CQM with important component information were established.

Next section presents more details about these changes.

### 5.1.1 Changes in relation to ISO/IEC 9126

Table 3 summarizes the changes that were performed in relation to ISO/IEC 9126. The characteristics and sub-characteristics that are represented in bold were not present in ISO/IEC 9126. They were added due to the need for evaluating certain CBSD-related properties that were not covered on ISO/IEC 9126. The sub-characteristic that is crossed was present in ISO/IEC 9126, but was removed in the proposed model. Finally, the sub-characteristic in italics had its name changed.

Table 3. Changes in the Proposed Component Quality Model, in relation to ISO/IEC 9126.

| **Characteristics** | **Sub-Characteristics** |
|---|---|
| Functionality | Suitability<br>Accuracy<br>Interoperability<br>Security<br>Compliance<br>**Self-contained** |
| Reliability | Maturity<br>Recoverability<br>Fault Tolerance |
| Usability | Understandability<br>**Configurability**<br>Learnability<br>Operability |
| Efficiency | Time Behavior<br>Resource behavior<br>**Scalability** |
| Maintainability | ~~Analyzability~~<br>Stability<br>Changeability<br>Testability |
| Portability | *Deployability*<br>Replaceability<br>Adaptability<br>**Reusability** |
| **Marketability** | **Development time**<br>**Cost**<br>**Time to market**<br>**Targeted market**<br>**Affordability**<br>**Licensing** |

The *Self-contained* 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 demand of data of his/her application.

Still on, the reason that software factories have adopted component-based approaches to software development is the premise 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-contained:** 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;
- **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 role are interlaced with interface code or *SQL* code, among others points; and

Additionally, one sub-characteristic was removed in order to adequate the model to the component context (represented in italic). In the *Maintainability* characteristic, the *Analyzability* sub-characteristic disappeared. Its main concern, according to ISO 9126, 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 a lot of 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), does 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 component than for software products. A little adaptation is necessary to bring the ISO 9126 characteristics definition to the component context.

Besides concentrating on quality characteristics only, another characteristic level was created, called *Marketability* (last row of Table 3). This characteristic presents some sub-characteristics that are important to a software component certification process, such as:

- **Development time:** The time consumed to develop a component;
- **Cost:** 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;
- **Affordability:** How affordable is the component; and

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

This information are not important to evaluate component quality, but are important to analyze some factors that bring credibility to the component customers (i.e. developers and designers), for example, the time that component was available to the market means that the component is more mature, because bugs are corrected and test cases were applied; the kind of component license is interesting for the costumer analyzed the cost/benefit of buy the component; and the target market of a component describes which domains a certain component can be applied.

### 5.1.2 Quality characteristics that were extended from ISO/IEC 9126

The previous section presented the major changes, in relation to ISO/IEC 9126, that were introduced in the proposed component quality model. This section presents the quality characteristics from ISO/IEC 9126 that were maintained in the proposed model, with some adaptation to better reflect the CBSD scenario. These are described next:

**Functionality:**

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

**Reliability:**

- **Maturity:** This characteristic evaluate 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 characteristic indicates whether the component can handle error situations, and the mechanism implemented in that case (e.g. exceptions); and
- **Fault Tolerance:** This characteristic indicates whether the component can maintain a specified level of performance in case of faults.

**Usability:**

- **Understandability:** This characteristic measure the degree of easiness to understand the component (e.g. documentation, descriptions, demos, API's, tutorial, etc.);
- **Learnability:** This characteristic try to measure the time and effort needed to master some specific tasks (e.g. usage, configuration, administration of the component); and
- **Operability:** This characteristic measure the ease to operate a component and to integrate the component into the final system.

**Efficiency:**

- **Time Behavior:** This characteristic indicates the ability to perform a specific task at the correct time, under specified conditions; and
- **Resource behavior:** This characteristic indicates the amount of the resources used, under specified conditions.

**Maintainability:**

- **Stability:** This characteristic indicates the stability level of the component in preventing unexpected effect caused by modifications;
- **Changeability:** This characteristic indicates whether specified changes can be accomplished and if the component can easily be extended with new functionalities; and
- **Testability:** This characteristic measures the effort required to test a component in order to ensure that it performs to its intended function.

**Portability:**

- **Replaceability**: This characteristic indicates whether the component is "backward compatible" with its previous versions; and
- **Adaptability:** This characteristic indicates whether the component can be adapted to different specified environments.

### 5.1.3 Summary

Table 4 shows another classification for the proposed component quality model. According to the moment when a 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 *life-cycle* (that are discernable at component and/or component-based system development). However, the *Marketability* characteristics are not classified according these two classifications.

Table 4. The Proposed Component Quality Model, with the sub-characteristics being divided into two kinds: runtime and life-cycle.

| Characteristics | Sub-Characteristics (Runtime) | Sub-Characteristics (Life-cycle) |
|---|---|---|
| Functionality | Accuracy<br>Security | Suitability<br>Interoperability<br>Compliance<br>Self-contained |
| Reliability | Fault Tolerance<br>Recoverability | Maturity |
| Usability | Configurability | Understandability<br>Learnability<br>Operability |
| Efficiency | Time Behavior<br>Resource Behavior<br>Scalability | |
| Maintainability | Stability | Changeability<br>Testability |
| Portability | Deployability | Replaceability<br>Adaptability<br>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: **(i)** characteristics, **(ii)** sub-characteristics, **(iii)** attributes and **(iv)** metrics (Figure 8). 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 (ISO/IEC 9126, 2001).

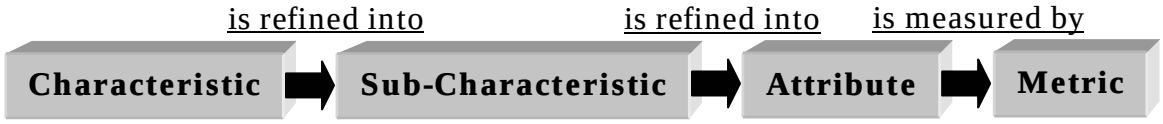


Figure 8. Relations among the quality model elements.

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 (ISO/IEC 9126, 2001).

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

## 5.2 Component Quality Attributes

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

First, it is necessary to define which kind of metrics will be used to measure the metrics used to measure each quality attributes. It was decided that for the attributes of the proposed component quality model, the following kinds of metrics would be adequate:

- **Presence:** This metric identifies whether an attribute is present in a component or not. It consists of a *boolean* value and a *string*. The *boolean* value is used to indicates whether the attribute is present and, if so, the *string* describes how the attribute is implemented by the component;
- **IValues:** This metric is used to indicate exact values of the component information. It is described by an *integer* variable and a *string* to indicates the unit (e.g. kb, mb, khz, etc.); and

- **Ratio:** This metric is used to describe percentages. It is measured by an *integer* variable with values between 0 and 100.

Table 5 shows the component quality attributes that are observable at *runtime*. The table groups the attributes by sub-characteristics, and indicates the metrics and kind of metrics used for evaluate each attribute.

Table 5. Component Quality Attributes for Sub-Characteristics that are observable at *Runtime*.

| **Sub-Characteristics (Runtime)** | **Attributes** | **Metrics** | **Kind of Metrics** |
|---|---|---|---|
| Accuracy | 1. Correctness | Tests results / precision | *Ratio* |
| Security | 2. Data Encryption | Mechanism implemented | *Presence* |
| | 3. Controllability | Number of interfaces / kind of controllability | *Ratio* |
| | 4. Auditability | Mechanism implemented | *Presence* |
| Recoverability | 5. Error Handling | Mechanism implemented | *Presence* |
| Fault Tolerance | 6. Mechanism available | Mechanism identification | *Presence* |
| | 7. Mechanism efficiency | Amount of errors tolerate / total errors found | *Ratio* |
| Configurability | 8. Effort for configure | Time spend to configure correctly | *IValues* |
| Time Behavior | 9. Response time | Time taken between a set of invocations | *IValues* |
| | 10. Latency<br>a. Throughput (“out”) | Amount of outputs produced with success / given period of time | *IValues* |
| | b. Processing Capacity (“in”) | Amount of inputs produced with success / given period of time | *IValues* |
| Resource Behavior | 11. Memory utilization | Memory used | *IValues* |
| | 12. Disk utilization | Disk used | *IValues* |
| Scalability | 13. Processing capacity | A large set of calls / response time of each call | *Ratio* |
| Stability | 14. Modifiability | A set of modifications, if possible / % of correct behavior | *Ratio* |
| Deployability | 15. Complexity level | Time taken for deploy | *IValues* |

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

**Accuracy Sub-Characteristic**

1. **Correctness:** This attribute evaluates the percentage of the results that were obtained with precision, specified by the user requirements;

**Security Sub-Characteristic**

2. **Data Encryption:** This attribute express the ability of a component to deal with encryption in order to protect the data it handles;
3. **Controllability:** This attribute indicates how the component is able to control the access to its provided interfaces;
4. **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**

5. **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**

6. **Mechanism available:** This attribute indicates the existence of fault-tolerance mechanisms implemented in the component;
7. **Mechanism efficiency:** This attribute measure the real efficiency of the fault-tolerance mechanisms that are available in the component;

**Configurability Sub-Characteristic**

8. **Effort for configure:** This attribute measures the ability of the component to be configured;

**Time Behavior Sub-Characteristic**

9. **Response time:** This attribute measures the time taken since a request is received until a response has been sent;
10. **Latency:**
    a. **Throughput (“out”):** This attribute measures the output that can be successfully produced over a given period of time;

b. **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**

11. **Memory utilization:** The amount of memory needed by a component to operate;
12. **Disk utilization:** This attribute specifies the disk space used by a component;

**Scalability Sub-Characteristic**

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

**Stability Sub-Characteristic**

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

**Deployability Sub-Characteristic**

15. **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 6. These attributes could be measured during the development of the component or the component-based system, by collecting relevant information for the model. The table also presents the metrics and kind of metric that is used to measure each attribute.

Table 6. Component Quality Attributes for Sub- Characteristics that are observable during *Life cycle*.

| Sub-Characteristics (Life cycle) | Attributes | Metrics | Kind of Metrics |
|---|---|---|---|
| Suitability | 1. Coverage | Specified functionality / % implemented | *Ratio* |
| | 2. Completeness | Implemented functionalities / total of specified functionalities | *Ratio* |
| | 3. Pre and Post-conditioned | Verification of the pre and post-conditions | *Presence* |

| | | | |
|---|---|---|---|
| | 4. Proofs of Pre and Post-conditions | Proofs verification | *Presence* |
| Interoperability | 5. Data Compatibility | Analysis of the data standard | *Presence* |
| Compliance | 6. Standardization | Implementation and documentation analysis | *Presence* |
| | 7. Certification | Verify documentation | *Presence* |
| Self-contained | 8. Dependability | Implementation analysis | *Ratio* |
| Maturity | 9. Volatility | Analysis of the time between commercial versions | *IValues* |
| | 10. Failure removal | Number of bugs fixed in a version | *IValues* |
| Understandability | 11. Documentation available | Documentation Analysis | *Presence* |
| | 12. Documentation quality | Documentation Analysis | *Presence* |
| Learnability | 13. Time and effort to (use, configure, admin and expertise) the component. | Component usage through implementation of some examples and demos | *IValues* |
| Operability | 14. Complexity level | All functionalities usage / time to operate | *Ratio* |
| | 15. Provided Interfaces | Number of provided interfaces | *IValues* |
| | 16. Required Interfaces | Number of required interfaces | *IValues* |
| | 17. Effort for operating | Operations in all provided interfaces / total of the provided interfaces | *Presence* |
| Changeability | 18. Extensibility | % of the functionalities that could be extended | *Ratio* |
| | 19. Customizability | Number of parameters to configure the provided interface / Number of interfaces | *Ratio* |
| Testability | 20. Test suit provided | Analysis of the test suites provided | *Presence* |
| | 21. Extensive component test cases | % of tests cases made / errors found/corrected | *Ratio* |
| | 22. Component tests in a specific environment | Number of environments that the component was tested | *IValues* |

| | | | |
|---|---|---|---|
| | 23. Proofs the components | Proofs Analysis | *Presence* |
| Adaptability | 24. Mobility | Number of containers that the component can be deployed | *IValues* |
| | 25. Configuration capacity | Effort needed to transfer a component to other environments | *Ratio* |
| Replaceability | 26. Backward Compatibility | Analysis of the compatibility with previous versions | *Ratio* |
| Reusability | 27. Domain abstraction level | Implementation and documentation analysis | *Ratio* |
| | 28. Architecture compatibility | Analysis of the component architecture | *Ratio* |
| | 29. Modularity | Packaging analysis | *Ratio* |
| | 30. Cohesion | Analysis of the inter-related parts | *Ratio* |
| | 31. Coupling | Analysis of the inter-related parts | *Ratio* |

A brief description of each quality attribute is presented next:

**Suitability Sub-Characteristic**

1. **Coverage:** This attribute measures how much of the required functionality is covered by the component implementation;
2. **Completeness:** It is possible that some implementations do not completely cover the services specified. This attribute measures the number of implemented operations compared to the total number of specified operations;
3. **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;
4. **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**

5. **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);

**Compliance Sub-Characteristic**

6. **Standardization:** This attribute indicates if the component conforms to international standards;
7. **Certification:** This attribute indicates if the component is certified by any internal or external organization;

**Self-contained Sub-Characteristic**

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

**Maturity Sub-Characteristic**

9. **Volatility:** This attribute indicates the average time between commercial versions;
10. **Failure removal:** This attribute indicates the number of bugs fixed in a given component version. The number of bugs fixed in a version could indicate that the new version is more stable or that the component contain a lot of bugs that will emerge;

**Understandability Sub-Characteristic**

11. **Documentation available:** This attribute deals with the component documentation, descriptions, demos, APIs and tutorials available, which have a direct impact on the understandability of the component;
12. **Documentation quality:** This attribute indicates the quality of the component documentation;

**Learnability Sub-Characteristic**

13. **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 usage, configuration, administration, or expertise the component);

**Operability Sub-Characteristic**

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

15. **Provided Interfaces:** This attribute counts the number of provided interfaces by the component as an indirect measure of its complexity;
16. **Required Interfaces:** This attribute counts the number of interfaces that the component requires from other components to operate;
17. **Effort for operating:** This attribute shows the average number of operations per provided interface (operations in all provided interfaces / total of the provided interfaces);

**Changeability Sub-Characteristic**

18. **Extensibility:** This attribute indicates the capacity to extend a certain component functionality (i.e. which is the percentage of the functionalities that could be extended);
19. **Customizability:** This attribute measures the number of customizable parameters that the component offers (e.g. number of parameters to configure each provided interface);

**Testability Sub-Characteristic**

20. **Test suit 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);
21. **Extensive component test cases:** This attribute indicates if the component was extensively tested before being made available to the market;
22. **Component tests in a specific environment:** This attribute indicates in which environments or platforms a certain component was tested;
23. **Proofs the components:** This attribute indicates if the component was formally tested;

**Adaptability Sub-Characteristic**

24. **Mobility:** This attribute indicates in which containers this component was deployed and to which containers this component was transferred;
25. **Configuration capacity:** This attribute indicates the percentage of the changes needed to transfer a component to other environments;

**Replaceability Sub-Characteristic**

26. **Backward Compatibility:** This attribute is used to indicate whether the component is "backward compatible" with its previous versions or not;

**Reusability Sub-Characteristic**

27. **Domain abstraction level:** This attribute measures the component abstraction level, related to its business domain;
28. **Architecture compatibility:** This attribute indicates the level of dependability of a specified architecture;
29. **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;
30. **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; and
31. **Coupling:** This attribute measures the coupling level of the components. A component should have low coupling in order to increase its reusability level.

The *Marketability* characteristics (composed of marketing characteristics) are statically measured through the information that is available in each component (e.g. description, documentation, web-site, etc.). This information is normally made available by the component vendor in the component market web-site. There is not a specific metric to measure these attributes. Instead, they are described by a *string* containing the information that was found (e.g. descriptions, price, component licensing, etc.).

Besides the quality and market characteristics presented, the model is complemented of other kind of characteristics. These characteristics bring relevant information for new customers and are composed of *Productivity*, *Satisfaction*, *Security* and *Effectiveness*. According to ISO 9126, these characteristics are called *Quality in Use* (ISO/IEC 9126-4, 2003). 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 expectation. These characteristics show whether the

developers or designers can trust in a component or not. Thus, *Quality in Use* characteristics are useful to show the component 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 the 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 characteristics.

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 were not included in the model because does not represent quality attributes for software component but contains relevant information's for a well-defined component certification process. These are presented next.

## 5.3 Other relevant Component Information

In an effective software component certification process, some kind of information is needed in order to complement the model with considerable information. Besides the quality characteristics presented early, other kind of information were identified: *Additional Information*.

The Table 7 shows the additional characteristics that were identified as being interesting to a software component certification 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 knows who is the responsible for that component, i.e. who maintains that component (e.g. a Tata's[1] component is more reliable than a component create by an unknown or a new software factory). Thus, it is identified the necessity of the *Organization Information*.

Table 7. Additional Information.

| | |
|---|---|
| **Additional Information** | Technical Information<br>• Component Version<br>• Programming Language<br>• Patterns Usage<br>• Lines of Code<br>• Technical Support<br>Organization Information<br>• CMM Level<br>• Organization's Reputation |

The *Additional Information* sub-characteristics are statically measured through the information that is available in each component (e.g. its description, documentation, web-site, etc.). This information is normally made available by the component vendor in the component market web-site. The characteristics are described by a *string* containing the information that was found (e.g. price, percentage, lines of code, etc.).

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.

## 5.4 Summary

This Chapter presented the proposed Component Quality Model, showed its characteristics and sub-characteristics, the quality attributes and its associated metrics. Some other relevant characteristics that were not included into the model because does not represent quality attributes for software component but contains relevant information's for a well-defined component certification process are also presented, in order to complement the CQM with important information that will be used in a well-defined software component certification process.

[1] Tata Consultancy Services
http://www.tcs.com

# A Formal Case Study

In order to determine whether the CQM meets its proposed goals, a formal case study was performed. This Chapter describes the steps of the formal case study and presents how these steps were performed in the study of this work.

The plan of the formal case study to be discussed follows the model proposed in (Wohlin et al., 2000) and the organization adopted in (Barros et al., 2002). The definition and the planning steps to be presented in the following sections are described in the future tense, symbolizing the precedence of the plan in relation to its execution.

This formal case study was meant to be as close as possible to an empirical study. It contemplates important points that a good empirical study should have, such as the previous planning and the precise definition of the null and alternative hypotheses to be evaluated. However, it cannot be considered a complete empirical study because it does not contemplate essential parts of an empirical evaluation, such as (Basili et al., 1986): Pilot Project, Qualitative Analysis, Internal Validity of the Study, External Validity of the Study, Validity of the Construction of the Study, among other points.

## 6.1 Component Quality Information Provided by Software Component Markets and by a Brazilian Software Factory – A Formal Case Study

### 6.1.1 Definition of the Formal Case Study

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

**Analyze** *the current gap between the required and provided information of the main component marketplaces and a Brazilian software factory*

**for the purpose of** *evaluating the component quality model*

**with respect** *to the efficiency of the model*

**from the point of view of the** *researchers, software and quality engineers*

**in the context of** *the software component certification area*.

### 6.1.2 Planning of the Formal Case Study

**Context.** The objective of the study is to evaluate how much of the information required for the metrics in the proposed CQM is currently available from the most widely used component markets. Thus, the main goal is to analyze the current gap between the *required* and *provided* information, and through this analysis the CQM will be refined and the metrics will become more realistic.

**Object of Study.** The objective of the study is evaluate the CQM usage, considering the software components of the most popular component markets from the Internet, such as *JARS*[1], *Flashline*[2] and *ComponentSource*[3], and the software components of a Brazilian software factory. These three markets were analyzed in order to establish a direction to get the components for evaluate. From these three sites, *JARS* and *Flashline* were finally discarded.

**JARS.** First, *JARS* offers mainly Java applets, and the information provided for each product is very scarce, basically a pointer to the Web site of the Applet developer (which usually provided few information). Thus, the JARS component market was discarded since none or a few of the quality information items could be obtained from its web site, and therefore it is not included in this study.

**Flashline.** *Flashline* contained, at the end of this study, over 9.000 components; however, the information provided together with the software components is mostly difficult to obtain and distill. Also, it was very hard to find the desired components. Furthermore, *ComponentSource* acquired *Flashline* marketplace in April 9, 2003, and all of its products were incorporated into *ComponentSource*'s offerings. For this reason, the *Flashline* component market

[1] http://www.jars.com
[2] http://www.flashline.com
[3] http://www.componentsource.com

was also discarded from this study. Therefore, the study will concentrate only on *ComponentSource*, which is the largest and most widely used software component market.

**ComponentSource.** *ComponentSource* organize its components in 94 different categories and contains some filters (e.g. programming language, best sellers, top reviews and new releases, among others) to support the software engineers in choosing its components according to their interests. Once a component is chosen, *ComponentSource* presents a page with the following sections:

- **Overview**: Presents an abstract of the component resources, its main functionalities, the programming language used and the component version;
- **Pricing and Licensing**: Lists the conditions of use and the price of each conditions (e.g. license for 1 developer, license for more than 10 developers, license for a server, among others);
- **Evals & Downloads**: Shows the assets (e.g. documentation, models, demos, API's, tutorial, etc) that the component owner make available in the site for demonstrations;
- **Compatibility**: Describes the component compatibilities, such as: platforms compatibility, application containers compatibility, resources compatibility, disk and memory space required, among other data;
- **Publisher Details**: A free space for the component owner to describe whatever he/she wants, for example, about his/her company and/or about the software product, etc;
- **Reviews**: This space describes the user's view (developer or designers) of the software product quality when it is used in a specific environment and in a specific context; and
- **Support Form**: The component owner can relate what kinds of support they provide to the users that buy the component.

Besides, the *ComponentSource* website contains more complete information about their products, making it easier for a software engineer to find a component, differently from other component marketplaces.

**A Brazilian software factory.** Currently, this company has about 520 employees, is ISO 9001 certified and is in preparation to obtain the CMM level 3. The company won the National Finep[4] Awards as the most Technologically Innovative Brazilian Institution in 2004, and the InfoExame[5] Award as the Best Brazilian Software Services Company in 2005. During seven years, this company has developed software for several domains, such as: supply chain, cell-phone games, business to business (B2B), government, among others.

**Subject.** The subject of the study will be a MSc. student at Federal University of Pernambuco and, system and quality engineers at the Brazilian software factory.

**Instrumentation.** The subject will act as a system and quality engineer, and will use Excel spread sheets (to collect specific information, i.e. runtime, life cycle, business, quality in use characteristics and additional information) in order to compute the component information found in the analyzed markets.

**Criteria.** The quality focus of the study demands criteria that evaluate the real efficiency of the model in measuring software components quality. This criteria will be evaluated quantitatively (amount of the component quality attributes found, amount of the component quality attributes measured and amount of the component information found in the market and not covered in the model).

**Hypothesis.** An important aspect of formal case studies is to know and to formally state what is going to be evaluated in the formal case study. A set of hypotheses was selected, as described next.

**- Null hypotheses, $H_0$:** these are the hypotheses that the experimenter wants to reject strongly. In this study, the null hypotheses determine that the CQM is not efficient to measure the component quality information that is available in the component marketplaces and in a software factory. According to the selected criteria, the following hypotheses can be defined:

[4] http://www.finep.gov.br
[5] http://www.infoexame.com.br

***Ho': amount of the component quality attributes of the model that were found in the market < 70%***

***Ho": amount of the component quality attributes of the model that could be measured < 70%***

***Ho"': amount of the component information that was found in the market and that is not covered in the model > 5%***

The market must contain information about most of the attributes that are described in the model. In this sense, null hypothesis $H_{0'}$ states that the amount of component quality attributes that can be found in the market is less than 70%. This study considers that an attribute is present in some market if a single component from that market contains information about that attribute.

However, if some attribute is only found in a few components, this means that the market is not providing the information needed by the component quality model, and the certification process can not be fully performed. In this way, null hypothesis $H_{0''}$ states that the amount of attributes that can be measured through information that is available in the market is less than 70%.

Also, ideally, no information that is important to quality assessment should be left out of the model. Since the proposed model was elaborated with basis on extensive experience from academy and industry, the expectation is that the amount of information found in the market and not covered by the model will be very close to this ideal situation. Thus, null hypothesis $H_{o'''}$ states that more than 5% of the information available at the markets is not covered by the model.

The values of these hypotheses (70%, 7-% and 5%, receptivity) were achieved through the feedback of some researchers of RiSE group and, software and quality engineers of a Brazilian software factory. Thus, these values constitute well-defined indices which the model must achieve in order to prove its viability.

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

***$H_1$: amount of the component quality attributes of the model that were found in the market >= 70%***

***$H_2$: amount of the component quality attributes of the model that could be measured >= 70%***

***$H_3$: amount of the component information that was found in the market and that is not covered in the model <= 5%***

**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).

### 6.1.3 Design of the Formal Case Study

**Data Used in the Study.** Fifty components were selected from *ComponentSource* (13 business components, 14 infra-structure components, 13 interface components and 10 database components). The components were chosen from *Best Sellers* and *Top Reviews* categories. The selection criterion was to choose only the most recommend and mostly used components. The idea was to evaluate the best components of the market, because such components would theoretically have more information that could be used to evaluate quality. However, this has proven not to be true, as described later in this Chapter.

Besides this criterion, no systematic approach to select the software components was applied. Of course, a more rigorous statistical analysis of the population could result in a more representative sample (maybe the best-sellers were better sold because they offered better information?). However, the main goal was to have enough coverage of the population for getting an initial impression of the component marketplace, even with some loss of precision. Future studies can extend this analysis in this direction.

From the Brazilian Software Factory, only six components were obtained. However, differently from those obtained from *ComponentSource*, which contained just a few samples and some documentation, these components were completely available (i.e. source code, documentation, models, samples, etc.). Again, no systematic approach was followed to select these components.

Among a set of 13,000 components of *ComponentSource*, 50 components were selected and, by the other hand, among a set of 20 component of a Brazilian software factory, 06 components were selected. Thus, the data obtained from this study can be used to make comparisons between the markets

because the data selected constitute approximately the same weight (50 from 13,000 and 06 from 20 components).

### 6.1.4 Instantiation of the Formal Case Study

**Selection of the Subjects.** For this study, no students were selected. Only one MSc. student participated in the formal case study.

**Random Capability.** The selection of the subjects for the formal case study was not random, since only one MSc. student accomplished the study.

**Analysis Mechanism.** To evaluate the hypotheses of the study, mechanisms of descriptive statistics (e.g. the mean), will be used.

### 6.1.5 Execution of the Formal Case Study

**Realization.** The formal case study was conducted as part of a MSc. Course in Software Engineering, during first semester in 2005, at the Federal University of Pernambuco. The formal study was executed over a period of 2 months, conducted by a single MSc. student. All the components were deployed, used (through a set of examples developed) and tested (using demos available and the examples developed) in the same environment. The computer used has Windows operating system, 1.8GHz of processor speed and 512mb of RAM.

### 6.1.6 Analysis of the Formal Case Study

**Data Collection.** In this study, the data collection was obtained from two sources (*ComponentSource marketplace* and a Brazilian software factory), as cited early. These two sources will be described next and, after that, the descriptive statistics and the concluding remarks are presented.

**1. Information Found in the *ComponentSource* marketplace.** The main difficulty that was found in this source is that the necessary information was not directly accessible. Sometimes there was almost no information available at all. This fact definitively rules out any possibility of trying to automate the search for this kind of component information. One must browse through files (e.g. word, acrobat reader, text, XML), manuals, documents (sometimes poor), models (a few models), or even demos (sometimes not runnable) in order to find the information that is necessary to evaluate components.

The main information found in the *ComponentSource* marketplace is composed of files that need to be "processed" in order to obtain the desired quality properties. Three kinds of files were provided for the components (the percentage represents the number of components that contained the information in relation to the total of components considered in the study):

- **Documentation (36,2%)**
  - **Help-Files (53%):** the information found into this kind of documentation is very simple and is often irrelevant to quality assessment purposes;
  - **API specifications (28%):** the majority of the API specifications found was related to Java, Java Beans and EJB components. Components developed with other technologies often do not present such documentation;
  - **User Documentation (32%):** only few of the components presented good quality user documentation. There is a lack of additional information that helps the developer in understanding the basic component functionalities;
  - **Tutorials (28%):** the quality of most tutorials (when present) is very interesting, considerably aiding the customer in the task of using the component;
  - **Manuals (40%):** these range from some publicity brochures to full-fledged 200 pages product manuals. However, sometimes the manuals are not helpful to understand the component functionalities, due to the lack of precise information about the components;
- **Demonstrations (92%):** demonstrations are not very helpful for automatically evaluating some of the quality attributes, although in most of the cases this is the only way to see how the product works. This fact is due to the component accessibility in the *ComponentSource*. The only way to completely access a component (i.e. source code, more documentation, support, etc.) is to buy it. Otherwise, the only information that is available are some demonstrations and usage examples, and;
- **UML Diagrams (8%):** the diagrams that were found show the structure of the component, together with some of its provided interfaces. These are very helpful for measuring some of the quality attributes defined in the

component quality model. However, only 8% of the components that were considered in this study contained such diagrams.

In order to present the results of the study, five tables are used: the attributes observed during runtime (Table 8), the attributes observed at life cycle (Table 9), the *Marketability* characteristics (Table 10), the *Additional Information* (Table 11) and the *Quality in Use* characteristics (Table 12).

Note that the effort required for distilling such information is not indicated (even though there were some difficult cases), but just whether it was possible or not to do so.

In the tables, those attributes that could not be identified in none of the sampled components due to the lack of information are represented with a "-". The attributes that were found in more than 70% of the components are shaded, in order to highlight the attributes to which the information was mostly found.

**Runtime.** One should notice, by looking at Table 8, the high number of the quality attributes with marks below 30%. In particular, the information for most of the runtime attributes was very difficult to obtain, especially for black-box components. This difficult can be seen in attribute "5. Error Handling". Only 20% of the components provided information regarding this attribute. In fact, this information was mostly found in those components for which a complete user manual was provided, and still it was very difficult to distill it.

It was also difficult to find information for attribute "1. Correctness". Although the measurement level that was achieved is considerably high when compared to the other quality attributes, sometimes the available examples did not work properly or/and did not exercise all of the component functionalities. In this way, the coverage of the services was not possible to assure. The good indices can be explained due to the relatively good examples that were available to test the component functionalities.

On the other hand, the information related to the static attributes, such as "11. Memory utilization" and "12. Disk utilization", was easier to find, mainly because these are the kind of quality attributes that component vendors are used to provided.

Table 8. Percentage of components whose attributes could be measured (at *runtime*) from the available information.

| Characteristics | Sub-Characteristics (Runtime) | Attributes | Measurable (%) |
|---|---|---|---|
| Functionality | Accuracy | 1. Correctness | 28% |
| | Security | 2. Data Encryption | 12% |
| | | 3. Controllability | 2% |
| | | 4. Auditability | 2% |
| Reliability | Recoverability | 5. Error Handling | 20% |
| | Fault Tolerance | 6. Mechanism available | 11% |
| | | 7. Mechanism efficiency | - |
| Usability | Configurability | 8. Effort for configure | 5% |
| Efficiency | Time Behavior | 9. Response time | - |
| | | 10. Latency | - |
| | | a. Throughput ("out") | - |
| | | b. Processing Capacity ("in") | - |
| | Resource Behavior | 11. Memory utilization | 88% |
| | | 12. Disk utilization | 100% |
| | Scalability | 13. Processing capacity | - |
| Maintainability | Stability | 14. Modifiability | - |
| Portability | Deployability | 15. Complexity level | - |

Table 9 presents the same phenomenon as Table 8, i.e. the relatively high number of the quality attributes with marks below to 30%.

Table 9. Percentage of components whose attributes could be measured (at *life cycle*) from the information available

| Characteristics | Sub-Characteristics (Life cycle) | Attributes | Measurable (%) |
|---|---|---|---|
| Functionality | Suitability | 1. Coverage | 24% |
| | | 2. Completeness | 24% |
| | | 3. Pre and Post-conditioned | - |
| | | 4. Proofs of Pre and Post-conditions | - |
| | Interoperability | 5. Data Compatibility | 37% |
| | Compliance | 6. Standardization | 2% |
| | | 7. Certification | 2% |

| | | | |
|---|---|---|---|
| | Self-contained | 8. Dependability | 26% |
| Reliability | Maturity | 9. Volatility | 21% |
| | | 10. Failure removal | 24% |
| Usability | Understandability | 11. Documentation available | 45,4% |
| | | a. User Doc. | 32% |
| | | b. API | 28% |
| | | c. Help System | 53% |
| | | d. Demos | 86% |
| | | e. Tutorial | 28% |
| | | 12. Doc. quality | 100% |
| | Learnability | 13. Time and effort to (use, configure, admin and expertise) the component. | - |
| | Operability | 14. Complexity level | 24% |
| | | 15. Provided Interfaces | 24% |
| | | 16. Required Interfaces | 24% |
| | | 17. Effort for operating | 14% |
| Maintainability | Changeability | 18. Extensibility | - |
| | | 19. Customizability | 7% |
| | Testability | 20. Test suit provided | - |
| | | 21. Extensive component test cases | 86% |
| | | 22. Component tests in a specific environment | 100% |
| | | 23. Proofs the components | - |
| Portability | Adaptability | 24. Mobility | 67% |
| | | 25. Configuration capacity | 14% |
| | Replaceability | 26. Backward Compatibility | 33% |
| | Reusability | 27. Domain abstraction level | 16% |
| | | 28. Crosscutting concerns level | - |
| | | 29. Architecture compatibility | - |
| | | 30. Modularity | 24% |
| | | 31. Cohesion | - |
| | | 32. Coupling | - |

**Life Cycle.** Some relevant quality attributes are below the 30% mark, such as “9. Volatility”, “10. Failure Removal”, “15. Provided Interfaces” and “16. Required Interfaces”. The first two characteristics are important to evaluate the maturity of the component and the other two are relevant for the customer to analyze the required and provided component services.

The better measurement percentages were achieved in “Understandability” (attributes 11 and 12), “Testability” (attributes 21 and 22) and “Adaptability” (attribute 24) sub-characteristics. All components contain some kind of documentation (good or bad) to present the component to the user. Although three quality attributes are bellow 32%, the measurement level is considered good and the documents satisfy, in parts, the information that is required for quality assessment.

Related to the “Testability” sub-characteristic, two quality attributes are extensively provided by the component vendors (“21. Extensive component test cases” and “22. Component tests in a specific environment”). These attributes are important to assure the correct component functionalities and to increase its reliability. Other quality attribute, the “24. Mobility”, also presented an elevated measurement level. This quality attribute is interesting to analyze the component behavior in different environments.

One point that should be stressed is that some static attributes, such as “13. Time and effort to Use, Configure, Admin and Expertise the component” and “17. Effort for Operating”, strongly depends on the level of expertise of the user. Also, such quality attributes cannot be measured by the component vendor or by any external independent quality assessor.

Most of the *Marketability* characteristics, according to Table 10, are available in the component market, except for one - the “1. Development Time” - that was not found in any component.

Other important information that was not found in some components is “3. Time to market”. Normally, the software factories state about its company, its history, etc. and are rare to found information about the component history.

Table 10. Percentage of components whose *Marketability* characteristics could be measured.

| Characteristics | Sub-Characteristics | Measurable (%) |
|---|---|---|
| Marketability | 1. Development time | 0% |
| | 2. Cost | 100% |
| | 3. Time to market | 45% |
| | 4. Targeted market | 100% |
| | 5. Affordability | 100% |
| | 6. Licensing | 100% |

All the components analyzed did not contain the characteristic "1. Development time" which, in some cases, is interesting for the customer knows how good the engineers that developed a component are.

Still on, the information shown in Table 11 is one of the most important to the model. These results are somehow surprising, because it was not expected that the components contained much information about these characteristics. On the other hand, this fact shows that the component market is growing since the past few years. Customers are interested in looking for components that aid in software development. The customer feedback is extremely important and should be stimulated.

However, the information found into the *ComponentSource* is not trustable, because it could be adapted for attract more attention for new costumer buy a certain component.

Table 11. Percentage of components whose *Quality in Use* characteristics could be measured.

| Characteristics | Sub-Characteristics | Measurable (%) |
|---|---|---|
| Quality in Use | 1. Productivity | 53% |
| | 2. Satisfaction | 64% |
| | 3. Security | 22% |
| | 4. Effectiveness | 58% |

Table 12 shows percentage of components whose Additional Information could be measured. This was by far the best results that were obtained in this study. Only one characteristic has little information: *Lines of Code*. This is due to the constant changes in software components (to correct bugs, to launch new versions with new functionalities, among other factors) and component vendors do not update this information often.

Table 12. Percentage of components whose *Additional Information* could be measured.

| Characteristics | Sub-Characteristics | Measurable (%) |
|---|---|---|
| Additional Information | Technical Information<br>• Component Version | 100% |
| | • Programming Language | 100% |
| | • Lines of Code | 5% |
| | • Technical Support | 100% |
| | Organization Information | 100% |
| | • CMM Level | 10% |
| | • Organization's Reputation | - |

**2. Information Found in the Brazilian Software Factory.** The study evaluated six Java-based, infra-structure components from the software factory. These components provided much more information than those found in *ComponentSource*. Not only the quality was greater, but also they provide several kinds of information that were not provided by *ComponentSource*. There were five kinds of information available, which are presented next. Again, the percentages regard to the number of components that provided the information in relation to all six components:

- **Documentation (53,2%):** the available documents are practical and contain guides for the developer to learn about the component usage – differently from those found in the *ComponentSource* marketplace.
  - **Help-Files (0%):** the selected components did not contain any kind of help-file;
  - **API specifications (100%):** the selected components were developed using the Java technology and, as occurred in the last study, all the components provided its API specification in JavaDoc format;
  - **User Documentation (66%):** many documents that contain interesting information to aid the developer in using the component were found, such as the component use guides (how to instantiate each component functionality, which are the component limitations and advantages, which environment the component properly work, etc.);
  - **Tutorials (0%)**: no tutorial was found among the selected components;

  - **Manuals (100%):** this was the most common document in the selected components. This is because in the Brazilian software factory, developers have the culture of providing a practical manual together with the component (i.e. the component architecture, the component interfaces and how to use it, the component scope, among other information) in order to aid the reutilization of such component by other developers. Also, the manuals that could be found in the Brazilian software factory contain more information required by the CQM than those found in *ComponentSource*;
- **Demonstrations (33%):** in this particular software factory, the focus is different from *ComponentSource*. Here, the idea is not to sell components to end-user customers, as black-boxes, but rather to provide self-encapsulated solutions that can be reused through different projects inside the same organization. Therefore, the "demonstrations" in the factory consist mostly in observing the component inside its final application, in a real execution environment. It must be stressed that this is a much more interesting way to perform the evaluation of the component. However, there were some cases where "stand-alone demonstrations" could be found;
- **UML Diagrams (43%):** the diagrams show the component and component-based application structure, together with some of its provided interfaces. The study identified that 66% of the components presented some kind of UML diagrams. However, not all of these were up to date in relation to the current component version, and did not provide helpful information for measuring quality attributes. Therefore, the real number of components that presented useful, up-to-date UML diagrams was 43%;
- **Source-Code (100%):** the main advantage of evaluating "*white-box*" components is due to the source code accessibility. There are some quality attributes that can only be measured by inspecting the source code; and
- **Component Developer/Designer (80%):** an interesting point to consider is that almost every time the software engineers that participated of the component development were available for consultation. Thus, whatever kind of information needed – even those that were not found in the documentations –could be obtained through the software engineers.

As in the previous case, the results were split in five tables: the attributes observed during runtime (Table 13), the attributes observed at life cycle (Table 14), the *Marketability* characteristics (Table 15), the *Additional Information* (Table 16) and the *Quality in Use* characteristics (Table 17). Still on, the effort required for distilling the information found in the documentation available was not indicated. In the tables, those attributes that could not be identified in none of the sampled components due to the lack of information are represented with a "-". The attributes that were found in more than 70% of the components are shaded, in order to highlight the attributes to which the information was mostly found.

**Runtime.** As shown in Table 13, the information available to evaluate software components are considerably greater in the software factory than in the analyzed component marketplace. This is mainly due to the availability of the source code and the presence of the software engineers/designers. The quality characteristics represented with 0% means that the information to measure such attributes exist, but the components do not contain these attributes.

Table 13. Percentage of components whose attributes could be measured (at *runtime*) from the information available.

| Characteristics | Sub-Characteristics (Runtime) | Attributes | Measurable (%) |
|---|---|---|---|
| Functionality | Accuracy | 1. Correctness | 100% |
| | Security | 2. Data Encryption | 0% |
| | | 3. Controllability | 0% |
| | | 4. Auditability | 0% |
| Reliability | Recoverability | 5. Error Handling | 100% |
| | Fault Tolerance | 6. Mechanism available | 0% |
| | | 7. Mechanism efficiency | 0% |
| Usability | Configurability | 8. Effort for configure | 100% |
| Efficiency | Time Behavior | 9. Response time | 100% |
| | | 10. Latency | 100% |
| | | a. Throughput ("out") | 100% |

| | | | |
|---|---|---|---|
| | | b. Processing Capacity ("in") | 100% |
| | Resource Behavior | 11. Memory utilization | 100% |
| | | 12. Disk utilization | 100% |
| | Scalability | 13. Processing capacity | 100% |
| Maintainability | Stability | 14. Modifiability | 100% |
| Portability | Deployability | 15. Complexity level | 100% |

**Life Cycle.** Related to the life cycle attributes (Table 14), the amount of information found is considerably good. Although 10 quality attributes were not able to be measured due to the lack of information, and 4 qualities attributes marks below to 45%, the amount of data available to evaluate software components is promissory.

As stated in Chapter 5, the formal quality attributes, such as "3. Pre-conditioned and Post-conditioned", "4. Proofs of pre-conditions and post-conditions" and "23. Proofs the components" would be very difficult to found in software components that do not relate to critical software, mainly because the formal approach is not widely adopted by the industry. This fact was confirmed by the study, which found no information for measuring these attributes.

Table 14. Percentage of components whose attributes could be measured (at *life cycle*) from the information available.

| **Characteristics** | **Sub-Characteristics (Life cycle)** | **Attributes** | **Measurable (%)** |
|---|---|---|---|
| Functionality | Suitability | 1. Coverage | 100% |
| | | 2. Completeness | 100% |
| | | 3. Pre and Post-conditioned | - |
| | | 4. Proofs of Pre and Post-conditions | - |
| | Interoperability | 5. Data Compatibility | 100% |
| | Compliance | 6. Standardization | - |
| | | 7. Certification | - |
| | Self-contained | 8. Dependability | 100% |
| Reliability | Maturity | 9. Volatility | - |
| | | 10. Failure removal | 33% |
| Usability | Understandability | 11. Documentation available | 42,2 |
| | | a. User Doc. | 66% |

| | | | |
|---|---|---|---|
| | | b. API | 100% |
| | | c. Help System | - |
| | | d. Demos | 45% |
| | | e. Tutorial | - |
| | | 12. Documentation quality | 100% |
| | Learnability | 13. Time and effort to (use, configure, admin and expertise) the component. | 98% |
| | Operability | 14. Complexity level | 100% |
| | | 15. Provided Interfaces | 100% |
| | | 16. Required Interfaces | 100% |
| | | 17. Effort for operating | 95% |
| Maintainability | Changeability | 18. Extensibility | 97% |
| | | 19. Customizability | 90% |
| | Testability | 20. Test suit provided | 33% |
| | | 21. Extensive component test cases | 33% |
| | | 22. Component tests in a specific environment | 100% |
| | | 23. Proofs the components | - |
| Portability | Adaptability | 24. Mobility | 80% |
| | | 25. Configuration capacity | 100% |
| | Replaceability | 26. Backward Compatibility | 66% |
| | Reusability | 27. Domain abstraction level | 100% |
| | | 28. Crosscutting concerns level | 100% |
| | | 29. Architecture compatibility | - |
| | | 30. Modularity | 100% |
| | | 31. Cohesion | 100% |
| | | 32. Coupling | 100% |

Additionally, new versions of the components are launched even if an error has occurred and, in this case, the “9. Volatility” of the launched versions

are misleading. A suggestion is to add this information into the *Release Notes* of the components (some *ComponentSource* components contain this information into their release notes).

It was difficult to find information regarding the *Understandability* sub-characteristic in the components of the software factory (attributes "11c. Help System" and "11e. Tutorial"). This is due to the fact that the documentation developed is meant to be practical, and less intended to marketing.

As expected, the *Marketability* characteristics are more easily found in *ComponentSource* marketplace than in the software factory (Table 15). This can be due to the difference of the context. In the *ComponentSource* the main concern is to sell the components, while in the software factory the components are meant to be internally used. In this sense, some information required to evaluate the *Marketability* characteristic were not available.

Table 15. Percentage of components whose *Marketability* sub-characteristics could be measured.

| Characteristics | Sub-Characteristics | Measurable (%) |
|---|---|---|
| Marketability | 1. Development time | - |
| | 2. Cost | - |
| | 3. Time to market | - |
| | 4. Targeted market | 100% |
| | 5. Affordability | 100% |
| | 6. Licensing | 100% |

The *Quality in Use* characteristics could be completely measured because the system engineers that build the systems with the support of the selected components were available (Table 16). Also, the feedback acquired here is more realistic because of the close contact with the people that actually use these components, differently from the *ComponentSource*.

Table 16. Percentage of components whose *Quality in Use* characteristics could be measured.

| Characteristics | Sub-Characteristics | Measurable (%) |
|---|---|---|
| Quality in Use | 1. Productivity | 100% |
| | 2. Satisfaction | 100% |
| | 3. Security | 100% |
| | 4. Effectiveness | 100% |

All the information required to evaluate *Additional Information* characteristics were completely available (Table 17). One should note that the difference between the software factory and *ComponentSource* is the presence of the "Lines of Code" attribute, which achieved 100% of measurement in the software factory.

Table 17. Percentage of components whose *Additional Information* could be measured.

| Characteristics | Sub-Characteristics | Measurable (%) |
|---|---|---|
| Additional Information | Technical Information<br>• Component Version | 100% |
| | • Programming Language | 100% |
| | • Lines of Code | 100% |
| | • Technical Support | 100% |
| | Organization Information | 100% |
| | • CMM Level | 100% |
| | • Organization's Reputation | - |

**3. Descriptive Statistics.** Once the necessary information was collected, the analysis could be performed. The collected data was grouped according to the two studies (*ComponentSource* and Brazilian software factory), as shown in Table 18. The percentages for null hypothesis H0' were calculated as the percentages of attributes that were found in the market in relation to all attributes of the model. The percentages for null hypothesis H0'' were calculated as the mean of all percentages of measurability (the last column – measurable – of the previous tables) for each market. Finally, the 0% values for null hypothesis ***H0'''*** were observed during the execution of the study. The standard deviation was not considered due to the small number of selected components.

As shown in Table 18, in the case of the Brazilian software factory, the results have shown that the null hypotheses were rejected, while in the case of *ComponentSource* marketplace, the second null hypothesis was not rejected:

***Ho': amount of the component quality attributes of the model that were found in the market < 70%***

***Ho'': amount of the component quality attributes of the model that could be measured < 70%***

***Ho''': amount of the component information that was found in the market and that is not covered in the model > 5%***

Table 18. Mean of the data from the study.

| | Mean | |
|---|---|---|
| | ***ComponentSource*** | ***Software Factory*** |
| **H0'** | 72,58% | 83,87% |
| **H0"** | 32,92% | 75,27% |
| **H0"'** | 0% | 0% |

i. **Amount of the component quality attributes found:** The CQM contains 62 quality attributes. In the study, 45 quality attributes (72,58%) were obtained from the information found in the *ComponentSource* marketplace, while in the Brazilian software factory this number was 52 (83,87%). These results reject hypothesis $H_{0'}$, which validates alternative hypothesis ***H1: amount of the component quality attributes of the model that were found in the market >= 70%.*** This implies that the quality attributes that compose the model is compatible with the component information found in the component market and in the repository of the software factory;

ii. **Amount of the component quality attributes measured:** During the study accomplished in the components available from the Brazilian software factory, 75,27% of the quality attributes required to the CQM was measured. In the case of the Brazilian software factory, null hypothesis $H_{0"}$ was rejected, validating hypothesis ***H2: amount of the component quality attributes of the model that could be measured >= 70%.*** However, only 32,92% of the information required to the CQM could be measured into the *ComponentSource* marketplace. Therefore, in the case of *ComponentSource*, the null hypothesis $H_{0"}$ was not rejected. This can be explained by the lack of information (source code, documents, models, tutorials, API's, information about the interfaces, information about components, etc) that could be freely accessed without buying the component, which was the approach taken by this study. However, in a real certification environment, these information would be available, and this number would probably be closes to the number achieved in the case of the Brazilian software factory (75,27%); and

iii. **Amount of the component information found in the market and not covered in the model:** Among all the component information found in the *ComponentSource* marketplace and in the Brazilian software

factory, everything was covered by the proposed component quality model, i.e. the study did not find any information that could not fit into some attribute of the model. In this sense, null hypothesis $H_{0'''}$ was rejected, validating hypothesis ***H3: amount of the component information that was found in the market and that is not covered in the model <= 5%.*** This indicates that the component characteristics that are defined in the CQM comprise a good set of characteristics, being a good candidate for software components evaluation.

**4. Conclusions.** The results shown in Table 18 confirm the existing gap between the information required by the "theoretical" metrics defined in current component quality models (compiled in the model proposed in this dissertation), and the information that is currently supplied by software components vendors and software factories that develop components. Even in the best case (the Brazilian software factory), almost 25% of the attributes could not be measured, and almost 16% of the attributes were not even present.

Another relevant point that was observed during this study is the gap between the information available in "black-box" components (those selected from *ComponentSource*) and in "white-box" components (those selected from a software factory). Although in some degree this was caused because this study did not actually buy the analyzed components, and therefore did not have access to more documents, it was observed that the source code is an important asset to be considered in component certification and must be available for a real component certification environment.

In order to improve this situation, component vendors should improve the available information about the components (both in quality and quantity) in order to aid the measurement task. Another improvement that could provide some benefits is to better structure and organizes the information aiming to facilitate the automation of the assessment processes as much as possible. Source code should be provided in order to maximize the amount of attributes that can be measured.

Still on, the quality characteristics and quality attributes of the model deserve further analysis, in order to refine the model in future evaluations. One example is the mean of attributes of the model that could be measured with

information from the market. In this study, the amount of attributes that could be measured was 32,92% and 75,27% (for *ComponentSource* and the Brazilian software factory, respectively). The number for *ComponentSource* is extremely low, indicating a large gap between the information from the model and information from the market.

However, this calculation considered the amount of attributes measured in relation to ALL attributes from the model, including those that could not be found, which implied that this mean would never be greater than the amount of attributes that could be found. For example, if 70% of the attributes were found in the market, and EVERY component presented information to measure 100% of these attributes, the calculated mean would still be 70%.

A more representative result could be obtained by excluding from this calculation the attributes that could not be found in ANY component. With this in mind, another hypothesis could be elaborated: ***H2´: AMONG THE ATTRIBUTES THAT COULD BE FOUND, the amount of attributes of the model that could be measured >= 70%.*** Recalculating the means for *ComponentSource* and the Brazilian software factory, these values arise to 45,36% and 89,75%, respectively, better reflecting the measurability of the attributes. However, the gap between the information from components obtained from *ComponentSource* and from the Brazilian software factory is still a large one.

## 6.2 Summary

This Chapter presented a formal case study, which analyzed components from a component market and from a Brazilian software factory, in order to analyze the gap between the *required* and *provided* information, and the completeness of the proposed component quality model. This study was accomplished with the intention of evaluate the CQM proposed in this dissertation and prove its viability to measure software components.

The study shows that software component quality is important to the software component market and the CQM proposed need to be revised in conjunction with the industry in order to acquire maturation and trust on it, mainly regarding to the assessment of “black-box” components. It also showed that the market do not contain all information needed by the component quality model, with almost 25% of the attributes being left un-measured.

# 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. A risk of selecting a product with unknown quality properties is no longer acceptable and, when happened, may cause catastrophic results (Jezequel et al., 1997). Thus, the 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 component quality model is strictly necessary. Thus, this dissertation presented a Component Quality Model, its characteristics, sub-characteristics, quality attributes and the related metrics that compose the model. As shown in Chapter 5, these four elements are essential for defining a quality model for whatever kind of software, in this case, software components.

A formal case study was defined, planned, executed and analyzed, showing the viability of the component quality model proposed. The study was accomplished with the world's largest component market and a Brazilian software factory, in order to analyze the gap between the information that is *required* by the model and the information that is *provided* by the market.

Some components were analyzed and a first evaluation of the model was possible.

The main goal of this research is to demonstrate that component certification 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 component quality model.

## 7.1 Research Contributions

The main contributions of this work could be split on three main aspects: **(i)** the realization of a survey related to the state-of-the-art in software component certification research; **(ii)** the proposition of a Component Quality Model to evaluate the software component quality; and **(iii)** the accomplishment of a formal case study, in order to evaluate the proposed component quality model.

- **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 component certification framework, aiming the development of a consistent and efficient component certification process;
- **The Component Quality Model.** The survey showed that software component quality is important to the component market and a quality model for software component is really necessary. In order to supply this necessity, a quality model for software components was defined, analyzing which quality characteristics and attributes are adequate for components and which metrics are useful for measuring these attributes; and
- **A Formal Case Study.** In order to determine whether the quality model meets its proposed goals, a formal case study was performed. This study analyzed the viability of the proposed component quality model,

identifying its main drawbacks. In this sense, a preliminary evaluation of the model was accomplished and future evaluations have been planned.

## 7.2 Related Work

Some related works could be found in the literature during this research. In this section, some works are presented, showing the main differences and similarities in relation to this work.

### 7.2.1 Meyer

In (Meyer, 2003), Meyer defines a direction, called “low road”, which leads to the qualification of existing components (e.g. defining a component quality model, determining the main characteristics for the component to achieve a certain level of quality). Meyer was concerned in establishing the main requirements that a component must have, in crescent order of importance. 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 - contains five categories with certain properties in each of these categories. Once all properties in one category are achieved, the component obtains a certain quality level. The component characteristics of this model were defined through the experiences of the Meyer’s research group in component technology and in CBSD.

### 7.2.2 Bertoa

In (Bertoa et al., 2002), Bertoa proposes a quality model for CBSD based on ISO 9126, which defines a set of quality attributes and their associated metrics for the effective evaluation of COTS components. Although the model is based on ISO 9126, Bertoa do not comment why some characteristics were removed and why other characteristics were added to the model. Even so, this model is one of the most well-defined in the literature; however, the motivation for defining the characteristics of the model is unknown.

### 7.2.3 Goulão

In (Goulão et al., 2002b), Goulão presents a quality model for components that are based on ISO 9126. The intention is to analyze the *black-box* and *white-box* characteristics of the software components in order to identify the characteristics that are essential to each kind of software components. After proposing such model, the author has concerned on defining some metrics for

measuring its characteristics. However, such metrics are still under development. Still on, Goulão did not mention anything related to the quality attributes of each characteristic. Some effort is also necessary to conclude this model.

### 7.2.4 Simão

In (Simão et al., 2003), Simão presents a quality model and a guide that can help in identifying and documenting the quality level of general software components. The proposed model presents a set of quality characteristics and sub-characteristics for software components based on ISO 9126 standard. Simão presents a large set of sub-characteristics for a quality model. From this set, a quality guide for software components was proposed, based on a field research accomplished with developers of components and component-based applications. On the other hand, Simão did not define the quality attributes and the metrics for evaluating software components.

### 7.2.5 Considerations about 7.2.1 – 7.2.4

The models above were not evaluated neither academic nor industrial scenarios: theirs real efficiency to evaluate the quality of software components is thus unknown. Additionally, Goulão and Simão did not specify quality attributes and such metrics that should be used to measure the quality characteristics proposed in the models above, making it difficult to use theirs in other scenarios.

Compared to the models above, the component quality model proposed in this dissertation will be applied, evaluated and tested in some Brazilian software factories that participate in the project described in Chapter 1. Thus, the model can become more efficient to solve the necessities of the component market (Heineman et al., 2001). Still on, the model will be evaluated at each four months by Brazilian software factories in order to correct some divergences that may be found. Besides this contribution, the proposed model contains some relevant characteristics that are not found in other models, such as *Marketability* characteristic, *Additional Information* and *Quality in Use*. In this way, the model is able to support the marketing characteristics (*Marketability*) and some other information that are useful for a certification process

(*additional information* and *quality in use*), which is not covered in those models.

## 7.3 Future Work

Through the results that were obtained (the survey of the state-of-the-art on software component certification area, the proposed component quality model and the formal case study), some directions stands up:

- **Software Component Maturity Model (SCMM).** Based on the proposed Component Quality Model, a SCMM will be constituted of certification levels where the components could be certified. The intention is to develop a model in which a component could increase its level of reliability and quality as it evolves (the SCMM is based on the same CMM principles (Paulk et al., 1993)). Thus, each customer decides which level is better for certifying its components, analyzing the cost/benefits of each level. The closer to the last level, the higher is the probability that the component is trusty and easily reused;
- **Software Component Certification framework.** Through the software component certification survey that was accomplished, a software component certification framework was defined. The intention is to investigate the component certification area in order to develop a robust and consistent component certification process. This framework contains four modules: **(i) a Component Quality Model**, with the purpose of determining which quality characteristics should be considered (defining the essential CBD characteristics) and which sub-characteristics are necessary; **(ii) a Certification Techniques Framework**, which determines certification techniques that will be used to evaluate the characteristics provided by the component quality model; **(iii) a Metrics Framework**, responsible for defining a set of metrics to track the properties of the components in a controlled way; and **(iv) a Certification Process**, responsible for defining a group of techniques and models to evaluate and certificate software components, aiming to establish a well-defined component certification standard. The main concern of this dissertation is based on the first module, however,

the other modules are currently being investigated (Alvaro et al., 2006b) by the RiSE group; and

- **Component Certification Center.** The long term plan could be to achieve a degree of maturity that could be used as a component certification standard for Software Factories, making it possible to create a Component Certification Center. Through the Brazilian projects that the RiSE group is involved to, such "dream" will become reality through the maturation of the process and the reliability of the software factories on that process.

## 7.4 Academic Contributions

The knowledge developed during the working remitted in the following publication:

- [Alvaro et al., 2006a] **Towards a Software Component Certification Process.** *Submitted to the 5th IEEE International Conference on COTS-Based Software Systems (ICCBSS), Florida, USA, 2006.*
- [Alvaro et al., 2006b] **A Software Component Quality Model.** *Submitted to the 28th ACM International Conference on Software Engineering (ICSE), Shanghai, China, 2006.*
- [Alvaro et al., 2005a] **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] **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] **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] **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., 2005a] **Key Developments in the Field of Software Reuse.** *Submitted to the Communications of the ACM, 2005.*
- [Almeida et al., 2005b] **A Survey on Software Reuse Processes**. In: The IEEE International Conference on Information Reuse and Integration (IRI), Las Vegas, Nevada, USA. IEEE Press. 2005.
- [Alvaro et al., 2005e] **Software Component Certification: A Component Quality Model**. In: Thehe 5th Brazilian Component-Based Development Workshop (WDBC), Short Paper, Juiz de Fora, MG, Brazil, 2005.
- [Almeida et al., 2004a] **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.

### 7.4.1 Other Publications

Besides the remits 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 in a M.Sc. level research.

- [Alvaro et al., 2005f] **Experiences with Software Reengineering**. In: The 21st International Conference on Software Maintenance (ICSM), Poster Session, Budapest, Hungary. IEEE Press. 2005.
- [Alvaro et al., 2005g] **ASPECT IPM: Towards an Incremental Process Model Based on AOP for Component-Based Systems**. In: The 7th International Conference on Enterprise Information Systems (ICEIS), Miami, USA. Lecture Notes in Computer Science (LNCS), Springer-Verlag. 2005.
- [Garcia et al., 2005] **Towards an Approach for Aspect-Oriented Software Reengineering**. In: The 7th International Conference on

Enterprise Information Systems (ICEIS), Miami, USA. Lecture Notes in Computer Science (LNCS), Springer-Verlag. 2005.

- [Lucredio et al., 2005] **Towards a Model-Driven Reuse Process**. In: The 31st IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Work in Progress Session, Porto, Portugal, 2005.
- [Almeida et al., 2004b] **Distributed Component-Based Software Development: An Incremental Approach**. In: The 28th IEEE Annual International Computer Software and Applications Conference (COMPSAC), Hong Kong. IEEE Press. 2004.
- [Garcia et al., 2004a] **Towards an effective approach for Reverse Engineering**. In: The 11th IEEE Working Conference on Reverse Engineering (WCRE), Poster Session, Delft University of Technology, Netherlands. IEEE Press. 2004.
- [Garcia et al., 2004b] **Using Reengineering and Aspect-Based Techniques to Retrieve Knowlegde Embedded in Object-Oriented Legacy System**. In: The IEEE International Conference on Information Reuse and Integration (IRI), Las Vegas, Nevada, USA. IEEE Press. 2004.
- [Alvaro et al., 2004a] **Towards an Incremental Process Model based on AOP for Distributed Component-Based Software Development**. In: The International Symposium on Distributed Objects and Applications (DOA), Poster Session, Larnaca, Cyprus. Lecture Notes in Computer Science (LNCS). Springer-Verlag. 2004.
- [Almeida et al., 2004c] **An Experimental Study about Distributed Component-Based Software Development**. In: The 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering, Short Paper, Madrid, Spain. 2004.
- [Almeida et al., 2004d] **Experiences with Component-Based Development**. In: The 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering, Poster Session, Madrid, Spain. 2004.

- [Almeida et al., 2004e] **Student's PLoP Guide: A Pattern Family to Guide Computer Science Students during PLoP Conferences**. In: The 4th Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP), Writers' Workshop, Porto das Dunas-CE, Brazil. 2004.

- [Garcia et al., 2004c] **Manipulating Crosscutting Concerns**. In: The 4th Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP), Writers' Workshop, Porto das Dunas-CE, Brazil. 2004.

- [Alvaro et al., 2004b] **Aspect IPM: Towards an Incremental Process Model based on AOP for Component-Based Development**. In: The First Brazilian Workshop on Aspect-Oriented Software Development, In Conjunction with the The 18th Brazilian Symposium on Software Engineering (SBES), Brasilia, Brazil. 2004.

- [Alvaro et al., 2004c] **Lições Aprendidas na criação de uma Fábrica de Software Open-Source**. In: The 5º Workshop de Software Livre (WSL). Porto Alegre-RS. 2004.

## 7.5 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 became 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 dissertation proposes a component quality model in order to establish the characteristics of a well-defined component certification process, and presents a formal case study, with the main software component markets and a Brazilian software factory, in order to analyze the gap between the required and provided information.

The study shows that software component quality is important to the component market, and that the quality model need to be revised in conjunction

with the industry in order to acquire maturation and trust on it. Thus, the intention is, in conjunction with the industry, to investigate the component certification area in order to develop: **(i)** a Component Quality Model, **(ii)** a Certification Techniques Framework, **(iii)** a Metrics Framework, and **(iv)** a Certification 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 certification process.

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

# 8 References


(Almeida et al., 2005a) Almeida, E. S.; Alvaro, A.; Meira, S. R. L. **Key Developments in the Field of Software Reuse**, *Submitted to the Communications of the ACM*, 2005.

(Almeida et al., 2005b) Almeida, E. S.; Alvaro, A.; Lucrédio, D.; Garcia, V. C.; Meira, S. R. L. **A Survey on Software Reuse Processes**. In: *The IEEE International Conference on Information Reuse and Integration (IRI)*, Las Vegas, USA, 2005.

(Almeida et al., 2004a) Almeida, E. S.; Alvaro, A.; Lucrédio, D.; Garcia, V. C.; Meira, S. R. L. **RiSE Project: Towards a Robust Framework for Software Reuse**, In: *IEEE International Conference on Information Reuse and Integration (IRI)*, Las Vegas, USA, pp. 48-53, 2004.

(Almeida et al., 2004b) Almeida, E. S.; Alvaro, A.; Lucrédio, D.; Prado, A. F.; Trevelin, L. C. **Distributed Component-Based Software Development: An Incremental Approach**. In: *The 28th IEEE Annual International Computer Software and Applications Conference (COMPSAC)*, Hong Kong, pp. 04-09, 2004.

(Almeida et al., 2004c) Almeida, E. S.; Lucrédio, D.; Alvaro, A.; Prado, A. F.; Trevelin, L. C. **An Experimental Study about Distributed Component-Based Software Development**. In: *The 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering (JIISIC)*, Short Paper, Madrid, Spain, 2004.

(Almeida et al., 2004d) Almeida, E. S.; Lucrédio, D.; Alvaro, A.; Prado, A. F.; Trevelin, L. C. **Experiences with Component-Based Development**. In: *The 4th Ibero-American Symposium on Software*

*Engineering and Knowledge Engineering (JIISIC)*, Poster Session, Madrid, Spain, 2004.

(Almeida et al., 2004e) Almeida, E. S.; Alvaro, A.; Lucrédio, D.; Garcia, V. C.; Piveta, E. K. **Student's PLoP Guide: A Pattern Family to Guide Computer Science Students during PLoP Conferences**. In: *The 4th Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP)*, Writers' Workshop, Porto das Dunas, Brazil, 2004.

(Alvaro et al., 2006a) Alvaro, A.; Almeida, E. S.; Meira, S. R. L. **Towards a Software Component Certification Process.** In: *The 5th IEEE International Conference on COTS-Based Software Systems (ICCBSS), Poster Session,* Florida, USA, 2006.

(Alvaro et al., 2006b) Alvaro, A.; Almeida, E. S.; Meira, S. R. L. **A Software Component Quality Model.** *Submitted to the 28th ACM International Conference on Software Engineering (ICSE)*, Shanghai, China, 2006.

(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.

(Alvaro et al., 2005e) Alvaro, A.; Almeida, E. S.; Meira, S. R. L. **Software Component Certification: A Component Quality Model**. In: *The 5th Brazilian Component-Based Development Workshop (WDBC), Short Paper,* Juiz de Fora, Brazil, 2005.

(Alvaro et al., 2005f) Alvaro, A.; Almeida, E. S.; Lucredio, D.; Garcia, V.C.; Prado, A.F. **Experiences with Software Reengineering**. In: *The 21st IEEE International Conference on Software Maintenance (ICSM)*, Poster Session, Budapest, Hungary, 2005.

(Alvaro et al., 2005g) Alvaro, A.; Almeida, E. S.; Lucrédio, D.; Garcia, V. C.; Prado, A. F.; Meira, S. R. L. **ASPECT IPM: Towards an Incremental Process Model Based on AOP for Component-Based Systems**. In: *The 7th International Conference on Enterprise Information Systems (ICEIS)*, Miami, USA. Lecture Notes in Computer Science (LNCS), Springer-Verlag, pp. 226-232, 2005.

(Alvaro et al., 2004a) Alvaro, A.; Lucrédio, D.; Prado, A. F.; Almeida, E. S. **Towards an Incremental Process Model based on AOP for Distributed Component-Based Software Development**. In: *The International Symposium on Distributed Objects and Applications (DOA),* Poster Session, Larnaca, Cyprus. Lecture Notes in Computer Science (LNCS). Springer-Verlag, PP. 38-39, 2004.

(Alvaro et al., 2004b) Alvaro, A.; Lucrédio, D.; Garcia, V. C.; Almeida, E. S. **Aspect IPM: Towards an Incremental Process Model based on AOP for Component-Based Development**. In: *The First Brazilian Workshop on Aspect-Oriented Software Development (WASP), In Conjunction with the The 18th Brazilian Symposium on Software Engineering (SBES)*, Brasilia, Brazil, 2004.

(Alvaro et al., 2004c) Alvaro, A.; Santos, T. L. V. L.; Andrade, P. R. P.; Vasconcelos, J. M. P.; Albuquerque, J.; Meira, S. R. L. **Lições Aprendidas na criação de uma Fábrica de Software Open-**

**Source**. In: *The 5º Workshop de Software Livre (WSL)*, Porto Alegre, Brazil, 2004.

(Arango, 1994) Arango, G. **A Brief Introduction to Domain Analysis**, In: *ACM symposium on Applied Computing*, USA, pp. 42-46, 1994.

(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., 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.

(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,** *Software Engineering Institute (SEI), Technical Report*, Vol. I, May, 2000.

(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.

(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 et al., 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.

(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.

(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.

(Clements et al., 2001) Clements, P.; Northrop, L. ***Software Product Line: Practices and Patterns***, SEI Series in Software Engineering, Addison Wesley, USA, 2001.

(CMMI, 2000) CMMI Product Development Team. **CMMI for Systems Engineering/Software Engineering/Integrated Product and Process Development/Supplier Sourcing, Version 1.1 Continuous Representation** (CMU/SEI-2002-TR-011, ESC-TR-2002-011). In: *Carnegie Mellon University/Software Engineering Institute (CMU/SEI)*, November, 2000.

(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.

(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, 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.

(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.

(Frakes et al., 1996) Frakes, W.; Terry, C. **Software Reuse: Metrics and Models**, In: *ACM Computing Survey*, Vol. 28, No. 02, pp. 415-435, 1996.

(Frakes et al., 1994) Frakes, W. B.; Isoda, S. **Success Factors of Systematic Reuse**, In: *IEEE Software*, Vol. 11, No. 05, pp. 15-19, 1994

(Gamma et al., 1995) Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. ***Design Patterns: Elements of Reusable Object-Oriented Software***. Addison-Wesley, 1995.

(Garcia et al., 2005) Garcia, V. C.; Lucrédio, D.; Prado, A. F.; Almeida, E. S.; Alvaro, A.; Meira, S. R. L. **Towards an Approach for Aspect-Oriented Software Reengineering**. In: *The 7th International Conference on Enterprise Information Systems (ICEIS)*, Miami, USA. Lecture Notes in Computer Science (LNCS), Springer-Verlag, pp. 274-279, 2005.

(Garcia et al., 2004a) Garcia, V. C.; Lucrédio, D.; Alvaro, A.; Almeida, E. S.; Prado, A. F. **Towards an effective approach for Reverse Engineering**. In: *The 11th IEEE Working Conference on Reverse Engineering (WCRE)*, Poster Session, Delft University of Technology, Netherlands, 2004.

(Garcia et al., 2004b) Garcia, V. C.; Lucrédio, D.; Alvaro, A.; Almeida, E. S.; Prado, A. F. **Using Reengineering and Aspect-Based Techniques to Retrieve Knowlegde Embedded in Object-Oriented Legacy System**. In: *The IEEE International Conference on Information Reuse and Integration (IRI)*, Las Vegas, USA, pp. 30-35, 2004.

(Garcia et al., 2004c) Garcia, V. C.; Alvaro, A.; Piveta, E. K.; Lucrédio, D.; Almeida, E. S.; Prado, A. F.; Zancanella, L. C. **Manipulating Crosscutting Concerns**. In: *The 4th Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP)*, Writers' Workshop, Porto das Dunas, Brazil, 2004.

(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.

(Heineman et al., 2001) Heineman, G. T.; Councill, W. T. ***Component-Based Software Engineering: Putting the Pieces Together***, Addison-Wesley, USA, 2001.

(Heineman et al., 2000) Heineman, G. T.; 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)*, Canada, pp. 661-664, 2000.

(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**, *NASA Software Technology Assurance Center (SATC)*, 1996.

(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 12119, 1994) ISO 12119, ***Software Packages – Quality Requirements and Testing***, International Standard ISO/IEC 12119, International Standard Organization (ISO), 1998.

(ISO/IEC 8402, 1994) ISO/IEC 8402, ***Quality management and quality assurance – Vocabulary***. International Standard ISO/IEC 8402, International Standard Organization (ISO), 1994.

(ISO/IEC 9000, 1994) ISO/IEC 9000, ***Management and Assurance Quality Standards***, International Standard ISO/IEC 9000, International Standard Organization (ISO), 1994.

(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.

(Keil et al., 2005) Keil, M; Tiwana, A. **Beyond Cost: The Drivers of COTS Application Value.** In: *IEEE Software*, Vol. 22, No. 03, pp. 64-69, 2005.

(Krueger, 1992) Krueger, C. W. **Software Reuse**, In: *ACM Computing Surveys*, Vol. 24, No. 02, pp. 131-183, 1992.

(Lim, 1994) Lim, W. C. **Effects of Reuse on Quality, Productivity, and Economics**, In: *IEEE Software*, Vol. 11, No. 05, pp. 23-30, 1994.

(Lucredio et al., 2005) Lucredio, D.; Fortes, R. P. M.; Alvaro, A.; Almeida, E. S.; Meira, S. R. L. **Towards a Model-Driven Reuse Process**. In: *The 31st IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, Work in Progress Session, Porto, Portugal, 2005.

(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.

(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) Mcllroy, 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***, 2th Edition Prentice Hall, London, 1997.

(Microsoft COM, 2005) Microsoft COM Technologies, At http://www.microsoft.com/com. Consulted in August 2005.

(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.

(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.

(OMG, 2005) Object Management Group (OMG), At http://www.omg.org. Consulted in August 2005.

(OMG CCM, 2002) Object Management Group (OMG), **CORBA Components**, Version 3, Document num. formal/02-06-65, June 2002.

(Paulk et al., 1993) Paulk, M.; Curtis, B.; Chrissis, M.; Weber, C. **Capability Maturity Model for Software, Version 1.1**, In: *Software Engineering Institute, Carnegie Mellon University*, CMU/SEI-93-TR-24, DTIC Number ADA263403, February, 1993.

(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, 2004) Pressman, R. ***Software Engineering: A Practitioner's Approach***. McGraw-Hill. 6th Edition. 2004.

(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.

(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.

(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.

(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***, 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.

(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.

(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***,

Building Reliable Component-Based Systems, I. Crnkovic, M. Larsson (editors), Artech House Publishers, July, pp. 29-37, 2002.

(Wallnau, 2005) Wallnau, K., **Software Component Certification: 10 Useful Distinctions**, *Technical Note CMU/SEI-2004-TN-031*, Available at: http://www.sei.cmu.edu/publications/documents/04.reports/04tn031.html, Consulted in August 2005.

(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.

(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***, Kluver 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 et al., 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.

Dissertação de Mestrado apresentada por **Alexandre Álvaro** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **"Software Component Certification: A Component Quality Model"**, 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. Sergio Castelo Branco Soares
Escola Politécnica / UPE

Prof. Silvio Romero de Lemos Meira
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 11 de outubro de 2005.

Prof. Nelson Souto Rosa
Vice-Coordenador da Pós-Graduação em
Ciência da Computação/UFPE