Towards a Software Component Quality Model

Alexandre Alvaro, Eduardo Santana de Almeida, Alexandre Marcos Lins de Vasconcelos, Silvio Romero de Lemos Meira
Federal University of Pernambuco and C.E.S.A.R – Recife Center for Advanced Studies and Systems, Brazil
{aa2, esa2, amlv, srlm}@cin.ufpe.br

Abstract

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 software component's industry reaches the maturity exhibited by other component industries. Problems such as the component selection by their integrators, the component catalogs formalization and the uncertain quality of third-party developed components, bring new challenges to the software engineering community. In this sense, this paper presents a brief survey on software component certification area, analyzing its lacks and further directions. Through this study, we propose a component quality model, describing consistent and well-defined characteristics for the components evaluation.

1. Introduction

One of the most compelling reasons for adopting component-based approaches to software development is the premise of reuse. The idea is to build software from existing components primarily by assembling and replacing interoperable parts. The implications for reduced development time and improved product quality make this approach very attractive [1].

Since components are reused in several occasions, they are likely to be more reliable than software developed from scratch, as they were tested under a larger variety of conditions. Cost and time savings result from the effort that would otherwise be necessary to develop and integrate the functionalities provided by the components in each new software application.

Most of the research dedicated to software components is focused on their functional aspects (i.e. component specification, component development, component tests, etc.). In our ongoing research, we are concerned with the evaluation of software components quality. This evaluation should be performed using a component quality model. However, there are several difficulties in the development of such a model, such as: **(1)** which quality characteristics should be considered, **(2)** how we can evaluate them and **(3)** who should be responsible for such evaluation [2].

However, 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 [3]. So, the common belief is that the market components are not reliable and this prevents the emergence of a mature software component market. Thus, the components market quality problems must be resolved to increase the reliability, and third-party certification programs would help to increase the trust in the market oriented components [4].

In this context, this paper describes the problems related to this new trend and discuss an initial direction in attempting to define a component quality model. Besides this introductory section, this paper is organized as follows. Section 2 presents a brief survey related to software component certification research. Section 3 proposes a component quality model, describing its main problems. Related approaches are considered in Section 4, and, finally, Section 5 presents the concluding remarks and directions for future work.

2. Component Certification: A Brief Survey

Existing literature is not that rich in reports related to practical software component certification experience, but some relevant research explores the theory of component certification in academic scenarios. In this sense, this section presents a brief survey of software component certification research, since the early 90's until today. More details about it can be seen in [5]. The timeline can be "divided" into two ages: from 1993 to 2001, where the focus was mainly on mathematical and; test-based models and, after 2001, where the focus was on techniques and models based in predicting quality requirements.

In 1993, Poore et. al. [6] developed an approach based on the use of three mathematics models (sampling, component and certification models), using test cases to report the failures of a system and, after, analyzing these data through mathematical methods in order to achieve a reliability index. Next, Wohlin & Runeson [7] presented the first method of component certification that consists in using modeling techniques, which make it possible not only to certify components, as well as to certify the system containing the components.

Two years later, in 1996, Rohde et al. [8] provided a synopsis of in-progress research and development in reuse and certification of software components at Rome Laboratory of the Air Force Material Command, Rome, NY. They developed a Certification Framework (CF) and defined the elements of the reuse context that are important to certification, to define the underlying models and methods of certification and, at last, to define a decision-support technique to construct a context-sensitive process for selecting the techniques and tools and applying them in order to certify components. After that, Sametinger [9] presented an interesting suggestion: the use of certification components levels. These levels depend on the nature, frequency, reutilization and importance of the component in a particular context. However, this is just a suggestion of certification levels and no practical work was actually done to evaluate it. Next, in 1998, the Trusted Components Initiative (TCI)[1] stands out. The TCI is a loose affiliation of researchers with a shared heritage in formal interface specification. Representative of TCI is the use of pre/post conditions on APIs [10]. This approach supports compositional reasoning, but only about a restricted set of behavioral properties of assemblies.

In this same year, Voas [11] defined a certification methodology using automated technologies, such as black-box (or COTS) testing and fault injection to determine whether the component fits into a specific scenario. Another work involving component tests may be seen in [12], where Wohlin & Regnell extended their previous research [7]. Now, focusing on techniques available for certifying both components and systems.

The state of the art, up to around 1999, was that components were being evaluated only with the results of the tests performed to the components. However, such testing had no well-defined way to measure the efficiency of the results. In 2000, Voas & Payne [13] defined some dependability metrics to measure the reliability of the components, and created a methodology for systematically increasing dependability scores by performing additional test activities.

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

In 2001, Morris et al. [14] proposed an entirely different model for software component certification. The model was based on test that developers supply in a standard portable form. So, the purchasers can establish the quality and suitability of purchased software.

Around 2001 some changes occurred in this area. The research started to change its focus and other issues began to be considered in component certification, besides testing, such as documentation quality, reuse level degree, among other properties.

Thus, in 2001, Stafford & Wallnau [15] developed 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

In this same year, Woodman et al. [16] analyzed some processes involved in various approaches to Component-Based Development (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*. Concomitantly, with the objective of obtaining the properties that a component should have, in 2003, Hissam et al. [17] introduced Prediction-Enabled Component Technology (PECT) as a means of packaging predictable assembly as a deployable product. A PECT is the integration of a component technology with one or more analysis technologies.

Other approach, in 2003, was proposed by McGregor et al. [18], defining a technique to provide component-level information to support prediction of assembly reliabilities based on properties of the components that form the assembly. Still, during 2003, a CMU/SEI's report [19] extended the Hissam et. al. work [17], describing how component technology can be extended in order to achieve Predictable Assembly from Certifiable Components (PACC). SEI's approach to PACC is Prediction-Enabled Component Technology (PECT).

In another work, in 2003, Meyer [20] highlighted the main concepts of trusted component along two complementary directions: a "low road" leading to qualification of existing components (e.g. defining a component quality model, determining the main characteristics of a component to achieve a certain level of quality), and a "high road" aimed at the production of components with fully proved correctness properties.

Moreover, two failure cases were found in the literature. The **first** failure occurred in the US government, when trying to establish criteria for certificating components [17], and the **second** failure happened with an IEEE committee, in an attempt to obtain a component certification standard [21].

By looking at these works, which represent the history and the current state-of-the-art in component certification, we may notice that this is a still immature area. Further research is needed in processes, methods, techniques, models, and tools, in order to obtain well-defined standards to component certification.

In general, the main certification idea's is bringing 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 and, in this way, increase the component market adoption. Normally, the software component evaluation occurs through models that evaluate its quality. These models describe and organize the component quality characteristics that will be evaluated. So, to measure the quality of a software component it is necessary to develop a quality model which represent the characteristics that will be considered to evaluate a component. Thus, we aim to investigate a Component Quality Model, identifying its characteristics and the sub-characteristics that compose the model.

3. Towards a Component Quality Model

3.1 The Motivation

According to [22], 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 software components. While some of their characteristics are appropriate to the evaluation of software components, others are not well suited for that task.

Even so, the software engineering community has expressed many and often diverging requirements to Component-Based Software Engineering (CBSE) and trustworthy components. A unified and prioritized set of CBSE requirements for trustworthy components is a challenge in itself [23]. Still, as cited early, 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 [2]. In this way, there is still no well-defined standard and component quality model to perform component certification [13, 14]. This fact is due also to the relatively novelty of this area [21].

Although recent, we found into literature some component quality models. The promising works are based on ISO 9126 [24]. 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 components and Component-Based Software Development.

Even so, the works found into literature try to analyze the ISO 9126 and propose such one model that are specific for software components [2, 22, 25]. The researchers aim to verify if each characteristics of ISO 9126 are adequate to the components context or if new characteristics need to be added into the model or removed. Thus, the component quality models were proposed based on the component technology and software quality experience of the researchers.

However, these models were not evaluated into academic or industrial scenario. In this way, the real efficiency to evaluate software components using these methods remains unknown. Additionally, two works [2, 25] did not specified the metrics that should be used to measure the characteristics proposed in the model.

3.2 The Model applied to the industry

As could be noted previously, still does not exist in the literature a well-defined component quality model that should be adopted. The most models were just theoretical proposes and were not applied an industrial or academic scale.

In this context, we are investigating effective ways to demonstrate that component certification is not only possible and practically viable, but also directly applicable in the software industry. And, 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 & Nascimento [26], 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[2], whose main concerns are: developing a robust framework for software reuse [27], in order to establish a standard to the component development; and defining and developing a repository system and a component certification process. This project has been developed in conjunction with the industry and academia (RiSE group[3] and other universities) in order to generate a well-defined model that will be capable of developing, evaluating quality, storing and, after that, making possible for software factories to reuse these components.

Given these motivations, a Software Component Certification framework is being investigated, with the objective of acquiring quality in software components that will be stored in repository systems. Basically, the framework that we intend to develop is composed of four

[2] http://www.finep.gov.br

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

modules (Figure 1): **(i) a Component Quality Model**, with the purpose of determining which quality characteristics should be considered and which sub-characteristics are necessary; **(ii) a Key CBD Quality Characteristics**, defining the essential CBD characteristics for an effective certification process in order to complement 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.

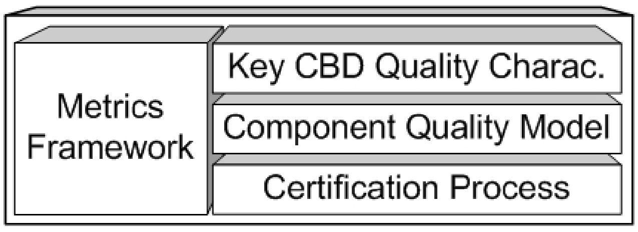

Fig. 1. Software Component Certification framework.

3.3 The Model proposal

Based on the project described previously, we are concerned on presenting a component quality model that will be capable of evaluating components of the software industry. The other elements of the framework will be discussed in future papers.

The component quality model proposed in this paper is based on ISO 9126 and some adaptations for components were accomplished. Still on, the model is composed of marketing characteristics which is not supported in other models. This model was discussed with the RiSE group member's, a PhD on software quality of Federal University of Pernambuco and with quality and software engineers that are specialists in component technologies of a software factory located in Recife, Brazil.

Additionally, we aim to adequate the model with the study accomplished through the current component market available in the internet (Flashline[4], Componentsource[5] and Jars[6]) [28]. This study showed which characteristics of the ISO 9126 could be measured through the information available in each component of these markets. And, even so, only a few characteristics could be completely measured, such as: suitability, changeability and resource behavior. This fact showed the difficulty of evaluating software components and the complexity of defining such a model to evaluate software components.

In this way, after analyzing this study and the ISO 9126, we developed the model. Table 1 shows the component quality model proposed, which is composed of six characteristics, as follows:

- **Functionality:** This characteristic express the ability of a software component to provide the required services and functions, when used under specified conditions;
- **Reliability:** This characteristics express the ability of the software component to maintain a specified level of performance when used under specified conditions;
- **Usability:** This characteristic express the ability of a software component to be understood, learned, used, configured, and executed, when used under specified conditions;
- **Efficiency :** This characteristic express 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
- **Business:** This characteristic express 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. As defined next, we identified some characteristics relevant to the component context, eliminated another characteristic that we think is not interesting to evaluate components, changed the name of one characteristic in order to adequate it to the component context, put another level of characteristics that contain relevant marketing information for a component certification process and established some characteristics that complement the component quality model with important component information's.

The new sub-characteristics identified are represented in bold. These sub-characteristics are added because we think necessary to evaluate certain properties that were not covered on ISO 9126. The *Self-contained* sub-characteristic is intrinsic of a component and must be analyzed.

The *Configureability* become essential to the developer analyze if the component can be easily configured. Thus, the developer verify the ability of configure a component in order to determine the complexity to deploy the component into a certain application.

[4] http://www.flashline.com
[5] http://www .componentsource.com
[6] http://www.jars.com

On the other hand, the *Scalability* is relevant to the model because express the capacity of the component to support major data volumes. So, the developer will know if the component support's the demand of data of his/her application.

Still on, the main concern that software factories has adopted the component technology is due to the fact that they can be reused. Thus, the *Reusability* sub-characteristics is very important to be considered in this model.

Table 1. Towards a Component Quality Model.

Characteristics	**Sub-Characteristics**
Functionality	Suitability Accuracy Interoperability Security Compliance **Self-contained**
Reliability	Maturity Recoverability Fault Tolerance
Usability	Understandability **Configureability** Learnability Operability
Efficiency	Time Behavior Resource behavior **Scalability**
Maintainability	Stability Changeability Testability
Portability	*Deployability* Replaceability Adaptability **Reusability**
Business	Development time Cost Time to market Targeted market Affordability

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

- **Self-contained:** The function that the component performs must be fully performed within itself;
- **Configureability:** The ability of the component be configurable (e.g. through a XML file or a text file, the number of parameters, among others);
- **Scalability:** The ability of the component to accommodate major data volumes without changing its implementation; and
- **Reusability:** The ability of the component be reused. This characteristic evaluate the reusability level through the abstraction level, if it is platform specific, if the business role are crosscutting with interface code or *sql* code, among others points.

Additionally, we removed one sub-characteristics in order to adequate the model to the component context. In the *Maintainability* characteristic, the *Analizability* sub-characteristic disappeared. In the context of components, we think that the result of the evaluation of this sub-characteristic will be insignificant, because a component is developed to attend certain functionalities of the application and, rarely are developed methods for its auto-analyze or to identify parts to be modified (which is the main concern of *Analizability* characteristic). For this reason, we tailored this sub-characteristic. Other component quality models [2, 22], removed this sub-characteristics too.

Concurrently, a sub-characteristic has changed its name and meaning in this new context. We identified just one sub-characteristic that should change its name, the *Installability*. Thus, we rename it as *Deployability*. After developed, the components are deployed (not installed) in an execution environment to make it possible their usage by others component-based applications that will be developed further. Through this modification, the understandability of this sub-characteristics become more clear 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 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.

Besides concentrating on quality characteristics only, we also created other characteristics level called *Business* (the name *Business* for this characteristic will be better analyzed and could be changed further). This characteristic presents some sub-characteristics that we think important to a certification process, such as:

- **Development time:** The time it takes to develop a component;
- **Cost:** The cost of the component;
- **Time to market:** The time it takes to make the component available on the market;
- **Targeted market:** The targeted market volume; and
- **Affordability:** How affordable is the component.

These information are not important to evaluate to quality of a component, but are important to analyze some factors that bring credibility to the component customers (i.e. developers and designers).

Still on, we identified some characteristics that bring relevant information for new customers, such as *Productivity*, *Satisfaction* and *Effectiveness*. According the ISO 9126, theses characteristics are called *Quality in Use* characteristics. This is the user's view (i.e. developer's or designer's) of the component, obtained when they use a certain component in a execution environment and analyze the results according their expectation. These characteristics show if the developers or designers can trust in a component. Thus, *Quality in Use* characteristics are useful to show the component behavior in different environments.

These characteristics are measured through the customer's feedback. In this way, the customers should be encouraged to buy a certain component that is well recommended.

Finally, as show in Table 2, we identified some additional characteristics that are interesting to the certification process. These characteristics are called *Considerable Information's* and are composed of: *Technical Information* and *Responsible*.

Table 2. Considerable Information's.

Considerable Information's	Technical Information • Component Version • Programming Language • Patterns Usage • Lines of Code • Technical Support Responsible

Technical Information is important to the developers analyze the actual state of the component (i.e. if the component has evolved, if any patterns was used in the implementation, etc.). It is composed of some elements, such as: *Component Version*, *Programming Language*, *Patterns Usage*, *Lines of Code* and *Technical Support*. Besides, it is interesting to the customer know who is the responsible for that component, i.e. who maintain that component. Thus, we identifying the necessity of the *Responsible* information.

The idea is that the *Considerable Information's* and *Business* characteristics should be "pre-requirements" to the component quality model proposed here. In this way, we think that the basic component information's are available and the model will complement these information's with the component evaluation.

Once we have discussed the general points added/changed/removed in the model, we will describe the other quality characteristics proposed for evaluating software components (excluding the characteristics described early), as follows:

Functionality:

- **Suitability**: This characteristic express 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, tutorials of the component);
- **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 an component and the ease 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 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.

Additionally, the moment in which a characteristic can be observed or measured also allows establishing another classification. Thus, the characteristics can be observable at *runtime* (that are discernable at component execution time) and observable during the product *life-cycle* (that are discernable at component and component-based systems development). However, the *Business* characteristic is not applied in this kind of representation because it is statically measured through the component information's. So, the Table 2 shows the component quality model classified into two classes.

Table 2. Component Quality Model.

Characteristics	**Sub-Characteristics (Runtime)**	**Sub-Characteristics (Life cycle)**
Functionality	Accuracy Security	Suitability Interoperability Compliance Self-contained
Reliability	Fault Tolerance Recoverability	Maturity
Usability	Configureability	Understandability Learnability Operability
Efficiency	Time Behavior Resource Behavior Scalability	
Maintainability	Stability	Changeability Testability
Portability	Deployability	Replaceability Adaptability Reusability

3.4 Component Quality Attributes

Once discussed the general points of the component quality model, in this section we will describe the quality attributes for measuring the sub-characteristics of components.

The metrics that will be used for measuring the attributes are the following:

- **Presence:** This metric identifies whether an attribute is present in a component or not. It consist 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 indicates exact values of the component information's. It is described by an integer variable and a string to indicates the unit (e.g. kb, mb, khtz, etc.); and
- **Ratio:** This metric is used to describe percentages. It is measured by an integer variable with values between 0 and 100.

Table 3 shows the quality attributes for components observable at *runtime* grouped by sub-characteristics and indicating the kind of metrics used.

Table 3. Component Quality Attributes for Sub-Characteristics measured at *Runtime*.

Sub-Characteristics (Runtime)	**Attributes**	**Metric**
Accuracy	1. Correctness	*Ratio*
Security	2. Data Encryption 3. Controllability 4. Auditability	*Presence* *Ratio* *Presence*
Recoverability	5. Error Handling	*Presence*
Fault Tolerance	6. Mechanism available 7. Efficiency	*Presence* *Ratio*
Configureability	8. Effort for configure	*Ratio*
Time Behavior	9. Response time 10. Latency a. Throughput ("out") b. Processing Capacity ("in")	*IValues* *IValues* *IValues*
Resource Behavior	11. Memory utilization 12. Disk utilization	*IValues* *IValues*
Scalability	13. Processing capacity	*Ratio*
Stability	14. Modifiability	*Ratio*
Deployability	15. Complexity level	*Ratio*

Now, a brief description of each quality attributes will be presented, as follows:

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

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 whether a component implements any auditing mechanism, with capabilities for recording users access to the system and to its data;
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);
6. **Mechanism available:** This attribute indicates the fault-tolerance mechanism implemented in the component;
7. **Efficiency:** This attributed measure the real efficiency of the fault-tolerance mechanism available in the component;
8. **Effort for configure:** This attribute measures the ability for the component to be configured;
9. **Response time:** This attribute measures the time taken since a request is received until a response has been sent;
10. **Latency:**
    - **Throughput ("out"):** This attribute measures the output that can be successfully produced over a given period of time;
    - **Processing Capacity ("in"):** This attribute measures the amount of input in-formation that can be successfully processed by the component over a given period of time;
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;
13. **Processing capacity:** This attribute measures the capacity of the component support a vast volume of data with the same implementation;
14. **Modifiability:** This attribute indicates the component behavior when accomplished some modification on it; and
15. **Complexity level:** This attribute indicates the effort for deploy a component in a specified environment.

Concomitantly, the quality attributes for components observable during *life cycle* are summarized in Table 4. These attributes could be measured during the component or component-based system development, collecting relevant information's for the model.

Table 4. Component Quality Attributes for Sub-Characteristics measured at *Life cycle*.

Sub-Characteristics (Life cycle)	**Attributes**	**Metric**
Suitability	1. Coverage 2. Completeness 3. Pre-conditioned and Post-conditioned 4. Proofs of pre-conditions and post-conditions	*Ratio* *Ratio* *Presence* *Presence*
Interoperability	5. Data Compatibility	*Presence*
Compliance	6. Standardization 7. Certification	*Presence* *Presence*
Self-contained	8. Dependability	*Ratio*
Maturity	9. Volatility 10. Failure removal	*IValues* *IValues*
Understandability	11. Documentation available 12. Documentation quality	*Presence* *Presence*
Learnability	13. Time and effort to (use, configure, admin and expertise) the component.	*IValues*
Operability	14. Complexity level 15. Provided Interfaces 16. Required Interfaces 17. Effort for operating	*Ratio* *IValues* *IValues* *Presence*
Changeability	18. Extensibility 19. Customizability	*Ratio* *Presence*
Testability	20. Test suit provided 21. Extensive component test cases 22. Component tests in a specific environment 23. Proofs the components	*Presence* *Presence* *Presence* *Presence*
Adaptability	24. Mobility 25. Configuration capacity	*Presence* *Ratio*
Replaceability	26. Backward Compatibility	*Presence*
Reusability	27. Domain abstraction level 28. Crosscutting concerns level 29. Architecture compatibility 30. Modularity	*Ratio* *Ratio* *Ratio* *Ratio*

In order to comprehend each quality attributes, a brief description is presented:

1. **Coverage:** This attribute tries to measure 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 tries to measure 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" requires and "what" provides.
4. **Proofs of pre-conditions and post-conditions:** This attribute indicates if the pre and post-conditions are formal proved in order to guarantee its correctness.
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);
6. **Standardization:** This attribute indicates the component conformance to international standards;
7. **Certification:** This attribute indicates if the component is certified by any internal or external organization;
8. **Dependability:** This attribute indicates if the component is not self-contained, i.e. if the component depend of other component to provide its specified services;
9. **Volatility:** This attribute measures the average time between commercial versions;
10. **Failure removal:** This attribute indicates the number of bugs fixed in a given version of the component. This number of bugs fixed in a version could indicates that the new version is more stable or that the component contain a lot of bugs that will emerge;
11. **Documentation available:** This attributes deal with the component documentation, descriptions, demos, and tutorials available, which have a direct impact on the understandability of the component;
12. **Documentation quality:** This attribute indicates the quality of the documents found into a component;
13. **Time and effort to (use, configure, admin and expertise) the component:** This attribute tries to measure the time and effort needed to master some specific tasks (such as usage, configuration, administration, or expertise the component);
14. **Complexity level:** This attribute indicates the capacity of the user 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 per provided interface (operations in all provided interfaces / total of the provided interfaces);
18. **Extensibility:** This attributes 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 he component offers (e.g. number of parameters to configure each interface provided);
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. performace);
21. **Extensive component test cases:** This attributes indicates if the component was extensive tested until be available to the market;
22. **Component tests in a specific environment:** This attributes indicates in which environments a certain component was tested;
23. **Proofs the components:** This attributed indicates if the component was formal tested;
24. **Mobility:** This attribute indicates which platforms this components was executed and which platforms the component was transferred ;
25. **Configuration capacity:** This attribute indicates the percentage of the changes needed to transferred a component to other environment;
26. **Backward Compatibility:** This attribute is used to indicating whether the component is "backward compatible" with its previous versions or not;
27. **Domain abstraction level:** This attribute measures the abstraction level of a component related to its specified business domain;
28. **Crosscutting concerns level:** This attribute indicates the code of the component, looking for analyze the code interlacement of business role, interface and SQL's;
29. **Architecture compatibility:** This attribute indicates the level of dependability of a specified architecture; and
30. **Modularity:** This attribute indicates the modularity level of the component, if it has modules, packages or all the source files are only grouped.

This section presented the initial version of the Component Quality Model, show its quality attributes and the its associated metrics. During the project (mentioned in section 3.2) this model can change in order to support the necessities of the software factories involved into the project and we attempt to capture the characteristics that will be really necessary to the model.

4. Related Work

Besides the works cited in section 3.1 [2, 22], in [20], Meyer define a direction, called "low road", leading to qualification of existing components (e.g. defining a component quality model, determining the main characteristics to component achieve a certain level of quality). Meyer was concerned in establishing the main requirements that the 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.

In [25], Simao & Belchior presented a quality model and a guide that can help 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. From this set, a quality guide for software components was proposed based on a field research accomplished with developers of components and component-based application.

Compared to the presented works, the component quality model proposed in this paper will be applied, evaluated and tested in some Brazilian software factories that participate in the project described earlier and with this, the model become more efficient to resolve the necessities of the component market [4]. Still on, the model will be evaluated at each four months by the Brazilian software factories in order to correct some divergences found in it. Besides this contribution, the model proposed contain some relevant characteristics that are not found in other models, such as *Bussines* characteristic, *Considerable Information's* and *Quality in Use*. In this way, the model is able to support the marketing characteristics.

5. Concluding Remarks and Future Directions

This work presented the state-of-the-art in software component certification research and proposed a initial component quality model in order to establish the requirements to a well-defined component certification process.

Our research group, in conjunction with the industry[7], aim to investigate the component certification area in order to: **(i)** establish a well-defined component quality model; **(ii)** define a framework (and corresponding metrics) to track the components properties; **(iii)** build a certification method and, finally, **(iv)** developed a structured component certification process.

The long term plan is, clearly, 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.

Currently, our research group is working with the definition of a Software Component Maturity Model (SCMM). Based on the Component Quality Model proposed, the SCMM will be constituted of certification levels where the components could be certified. The intention is to develop a model in which the component could increase its level of reliability and quality as it evolutes (the SCMM is based on the same CMM principles [29]). Besides, a set of metrics will be defined to track the properties of the components in a controlled way. This model and such metrics will be described in future papers.

6. Acknowledgements

This work is supported by C.E.S.A.R - Recife Center for Advanced Studies and Systems, Brazil, and, Financiadora de Estudos e Projetos (FINEP), Brazil. The members of RiSE group contributed with important insights for the initial analyses of the component quality model.

7. References

[1] C.W. Krueger, "Software Reuse", *ACM Computing Surveys*, Vol. 24, No. 02, June, 1992, pp. 131-183.

[2] M. Goulão, F.B. Abreu, "Towards a Component Quality Model", *Work in Progress Session of the 28th IEEE Euromicro Conference*, Dortmund, Germany, 2002.

[3] J.M. Jezequel, B. Meyer, "Design by Contract: The Lessons of Ariane", *IEEE Computer*, Vol. 30, No. 2, 1997, pp. 129–130.

[4] G.T. Heineman, W.T. Councill, "Component-Based Software Engineering: Putting the Pieces Together", *Addison-Wesley*, USA, 2001.

[5] A. Alvaro, E.S. Almeida, S.R.L. Meira, "A Software Component Certification: A Survey", *Submitted to the 31st IEEE Euromicro Conference*, Component-Based Software Engineering Track, 2005.

[6] J. Poore, H. Mills, D. Mutchler, "Planning and Certifying Software System Reliability", *IEEE Computer*, Vol. 10, No. 01, January, 1993, pp. 88-99.

[7] Currently, this company has about 380 employees and is in preparation to obtain the CMM level 3

[7] C. Wohlin, P. Runeson, “Certification of Software Components”, *IEEE Transactions on Software Engineering*, Vol. 20, No. 06, June, 1994, pp 494-499.

[8] S.L. Rohde, K.A. Dyson, P.T. Geriner, D.A. Cerino, “Certification of Reusable Software Components: Summary of Work in Progress”, *In Proceedings of the 2nd IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, Canada, 1996, pp. 120-123.

[9] J. Sametinger, “Software Engineering with Reusable Components”, *Springer Verlag*, USA, 1997.

[10] B. Meyer, “Object-Oriented Software Construction”, *2th Edition Prentice Hall*, London, 1997.

[11] J.M. Voas, “Certifying Off-the-Shelf Software Components”, *IEEE Computer*, Vol. 31, No. 06, June, 1998, pp. 53-59.

[12] C. Wohlin, B. Regnell, “Reliability Certification of Software Components”, *In the Proceedings of the 5th IEEE International Conference on Software Reuse (ICSR)*, Canada, 1998, pp. 56-65.

[13] J.M. Voas, J. Payne, “Dependability Certification of Software Components”, *Journal of Systems and Software*, Vol. 52, No. 2-3 , June, 2000, pp. 165-172.

[14] J. Morris, G. Lee, K. Parker, G. A. Bundell, C. P. Lam, “Software Component Certification”, *IEEE Computer*, Vol. 34, No. 09, September, 2001, pp 30-36.

[15] J. Stafford, K. C. Wallnau, “Is Third Party Certification Necessary?”, *In the Proceedings of the 4th ICSE Workshop on Component-Based Software Engineering (CBSE)*, Canada, May, 2001.

[16] M. Woodman, O. Benebiktsson, B. Lefever, F. Stallinger, “Issues of CBD Product Quality and Process Quality”, *In the Proceedings of the 4th ICSE Workshop on Component-Based Software Engineering (CBSE)*, Canada, May, 2001.

[17] S.A. Hissam, G.A. Moreno, J. Stafford, K.C. Wallnau, “Enabling Predictable Assembly”, *Journal of Systems and Software*, Vol. 65, No. 03, March, 2003, pp. 185-198.

[18] J.D. McGregor, J.A. Stafford, I.H. Cho, “Measuring Component Reliability", *In the Proceedings of the 6th ICSE Workshop on Component-Based Software Engineering (CBSE)*, USA, May, 2003, pp. 13-24.

[19] K.C. Wallnau, “Volume III: A Technology for Predictable Assembly from Certifiable Components”, *Software Engineering Institute (SEI), Technical Report*, Vol. 03, April, 2003.

[20] B. Meyer, “The Grand Challenge of Trusted Components”, *In the Proceedings of 25th International Conference on Software Engineering (ICSE)*, USA, 2003, pp. 660–667.

[21] M. Goulao, F. Brito e Abreu, “The Quest for Software Components Quality”, *In the Proceedings of the 26th IEEE Annual International Computer Software and Applications Conference (COMPSAC)*, England, August, 2002, pp. 313-318.

[22] M. Bertoa, A. Vallecillo, “Quality Attributes for COTS Components”, *In the Proceedings of the 6th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE)*, Spain, 2002.

[23] H. Schmidt, “Trustworthy components: compositionality and prediction”, *Journal of Systems and Software*, Vol. 65, No. 3, March, 2003, pp. 215-225.

[24] ISO 9126, “Information Technology – Product Quality – Part1: Quality Model”, *International Standard ISO/IEC 9126, International Standard Organization*, June, 2001.

[25] R.P.S. Simão, A. Belchior, “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.

[26] K.C. Weber, C.J. Nascimento, “Brazilian Software Quality 2002”, *In the Proceedings of 24th International Conference on Software Engineering (ICSE)*, EUA, pp. 634-638, 2002.

[27] E.S. Almeida, A. Alvaro, D. Lucrédio, V.C. Garcia, S.R.L. Meira, “RiSE Project: Towards a Robust Framework for Software Reuse”, *In IEEE International Conference on Information Reuse and Integration (IRI)*, USA, 2004.

[28] M.F Bertoa, J.M. Troya, A. Vallecillo, “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.

[29] M. Paulk, B. Curtis, M. Chrissis, C. Weber, “Capability Maturity Model for Software, Version 1.1”, *Software Engineering Institute*, CMU/SEI-93-TR-24, DTIC Number ADA263403, February, 1993.