

Quality Attributes for a Component Quality Model

Alexandre Alvaro, Eduardo Santana de Almeida, Silvio Romero de Lemos Meira
*Federal University of Pernambuco and C.E.S.A.R – Recife Center for Advanced Studies and Systems,
Brazil*

{alexandre.alvaro, eduardo.almeida, silvio}@cesar.org.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. By the other hand, the software components certification area is still immature and further research is needed in order to obtain well-defined standards for certification. In this way, we aim to propose a component quality model, describing mainly the quality attributes and related metrics 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 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, development, 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 and quality attributes 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 component market quality problems must be resolved to increase the reliability, and third-party certification programs would help acquire trust in the market oriented components [4].

In this context, this paper describes the problems related to this new trend and discusses 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 overview related to software component certification research. Section 3 proposes a component quality model, describing, mainly, its quality attributes and related metrics for evaluate the component quality. Section 4 presents the concluding remarks and directions for future work.

2. Component Certification: A Brief Overview

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. 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. More details about the component certification survey can be seen in [5].

By looking at the works covered in [5], 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 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. In this way, we aim to investigate a Component Quality Model, identifying its characteristics, the sub-characteristics, the quality attributes and the related metrics that compose the model.

3. The Component Quality Model

3.1 The Motivation

According to [6], 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 Component-Based Software Engineering (CBSE) and trustworthy components. A unified and prioritized set of CBSE requirements for trustworthy components is a challenge in itself [7]. Still, as cited early, there are several difficulties in the development of component quality model, such as **(i)** which quality characteristics and quality attributes 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 [8, 9]. This fact is due also to the relatively novelty of this area [10].

Although recent, we found into literature some component quality models. The promising works are based on ISO 9126 [11]. 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.

Even so, the works found into literature looking for analyze the ISO 9126 standard and propose such one model that are specific for software components [2, 6, 12]. The researchers aim to verify if each characteristics of ISO 9126 are adequate to the component context or if new characteristic need to be added or removed to the model. Thus, the quality models were proposed based on the component technology and software quality experience’s 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 models remains unknown. Additionally, two works [2, 12] did not specified the quality attributes and the metrics that should be used to measure the quality characteristics proposed in the model, becoming difficult to use this model in whatever scenario.

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 [13], 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 government¹, whose main concerns are: developing a robust framework for software reuse [14], 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

¹ <http://www.finep.gov.br>

conjunction with the industry and academia (RiSE group² 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 modules (Figure 1): **(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 determine the 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 certify software components, aiming to establish a well-defined component certification standard.

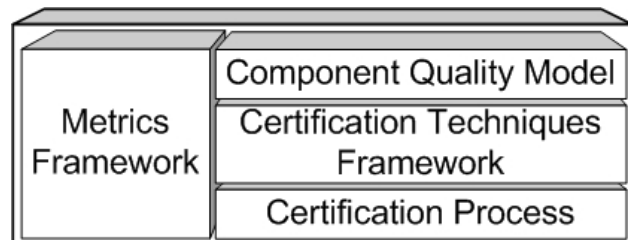


Figure 1. Software Component Certification framework.

3.2 The Model

Based on the project described previously, we are concerned on presenting the quality attributes of the component quality model, which 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 is based on ISO 9126 and some adaptations for components were accomplished. The model is composed of marketing characteristics and some relevant component information's which is not supported in other component quality models. The complete description of the component quality model, its characteristics and the other important characteristics, such as *Quality in Use* and *Considerable Information's*, can be seen into [15].

Table 1 show the component quality model classified into two classes: the quality characteristics that can be observable at *runtime* (that are discernable at component execution time) and the quality characteristics that can be observable during the product *life-cycle* (that are discernable at component and component-based systems development).

Table 1. 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	<i>Deployability</i>	Replaceability Adaptability Reusability

Table 1 shown the component quality model proposed, which is composed of six characteristics, as follows:

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

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

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

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

On the other hand, the *Scalability* is relevant because express the ability 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 reason that software factories have adopted component-based approaches to software development is the premise of reuse. In this sense, the *Reusability* sub-characteristics are so important to be considered too.

Additionally, we removed one sub-characteristic 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 (i.e. this is the main concern of *Analizability* characteristic, according to ISO 9126). For this reason, we tailored this sub-characteristic.

Concurrently, a sub-characteristic has changed its name and meaning in this new context. We identified just one sub-characteristic, represented in italic, which should change its name, the *Installability*. Thus, we rename it as *Deployability* (represented in italic). 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 developed further. Through this modification, the understandability of this sub-characteristic becomes 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*. This characteristic presents some sub-characteristics that we think important to a certification process, such as: (i) **Development Time**, the time it takes to develop a component; (ii) **Cost**, the cost of the component; (iii) **Time to Market**, the time it takes to make the component available on the market; (iv) **Targeted Market**, the targeted market volume; and (v) **Affordability**, how affordable is the component. These 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 some bugs are corrected and some test cases were applied.

3.3 Component Quality Attributes and related Metrics

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, presented in Table 1.

An *attribute* is a measurable physical or abstract property of an entity. By making a measurement, a measure is assigned to an attribute of an entity, using a metric. A *metric* is the defined measurement method and the measurement scale and the *measure* is the number or category assigned to an attribute [4].

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 indicate 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 2 shows the quality attributes for components observable at *runtime* grouped by sub-characteristics and indicating the kind of metrics used.

Table 2. Component Quality Attributes for Sub-Characteristics measured at *Runtime*.

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

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 (e.g. exceptions in Java);
6. **Mechanism available:** This attribute indicates the fault-tolerance mechanism implemented in the component;
7. **Mechanism efficiency:** This attribute 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 information 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 3. These attributes could be measured during the component or component-based system development, collecting relevant information’s for the model.

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

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

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 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*” the component requires and 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 the correctness of the component functionalities;

- 5. Data Compatibility:** This attribute indicates whether the format of the data handled by the component is compliant with any international standard or convention;
- 6. Standardization:** This attribute indicates if 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 component version;
- 11. Documentation available:** This attribute deal with the component documentation, descriptions, demos, API's 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 in certain 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 interface (operations in all provided interfaces / total of the provided interfaces);
- 18. Extensibility:** This attribute indicates the capacity to extend a certain component functionality;
- 19. Customizability:** This attribute measures the number of customizable parameters that the component offers;
- 20. Test suit provided:** This attribute indicates whether some test suites are provided for checking the functionality of the component;
- 21. Extensive component test cases:** This attribute indicates if the component was extensive tested until be 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 formal tested;
- 24. Mobility:** This attribute indicates which containers this components was deployed and which containers the component was transferred;
- 25. Configuration capacity:** This attribute indicates the percentage of the changes needed to transferred a component to other environments;
- 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 component abstraction level related to its business domain;
- 28. Crosscutting concerns level:** This attribute indicates if the component code is interlaced (e.g. business role code with interface code and SQL's code), becoming difficult its reusability;
- 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.

4. Conclusion and Future Directions

This work presented the quality attributes of the Component Quality Model proposed in [15], showing its associated metrics. During the project (mentioned in section 3.1) the characteristics, sub-characteristics and the quality attributes can change looking for support the necessities of the software factories involved into the project. We attempt to capture the characteristics that will be really necessary to the model in order to it become a stronger candidate to be a component certification standard to the industry.

Our research group, in conjunction with the industry³, aim to investigate the component certification area in order to: **(i)** define a component quality model, determining which quality characteristic should be considered and which sub-characteristics are necessary; **(ii)** determine the certification techniques that will be used to evaluate the characteristics provided by the component quality model; **(iii)** a framework for defining a set of

³ Currently, this company has about 520 employees and is in preparation to obtain the CMM level 3.

metrics to track the properties to the component in a controlled way; and (iv) build a certification process which is responsible for defining a group of techniques, methods and models to evaluate and certificate software components.

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 evolves (the SCMM is based on the CMM principles [16]).

Besides, our group has accomplished a preliminary evaluation with the component quality model proposed [17]. A number of “black-box” components from the component marketplaces (*Componentsource*⁴) and “white-box” components from a Brazilian software factory were selected to be analyzed. The SCMM model and the initial component quality model evaluation will be described in future papers.

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”, *In the 31st IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, Component-Based Software Engineering Track, 2005.
- [6] 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.
- [7] H. Schmidt, “Trustworthy components: compositionality and prediction”, *Journal of Systems and Software*, Vol. 65, No. 3, March, 2003, pp. 215-225.
- [8] 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.
- [9] 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.
- [10] 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.
- [11] ISO 9126, “Information Technology – Product Quality – Part1: Quality Model”, *International Standard ISO/IEC 9126*, *International Standard Organization*, June, 2001.
- [12] 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.
- [13] 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.
- [14] E.S. Almeida, A. Alvaro, D. Lucredio, 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.
- [15] A. Alvaro, E.S. Almeida, S.R.L. Meira, “Towards a Software Component Quality Model”, *Submitted to the 5th International Conference on Quality Software (QSIC)*, 2005.
- [16] 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.
- [17] A. Alvaro, E.S. Almeida, S.R.L. Meira, “Component Quality Information Provided by Software Component Markets and a Brazilian Software Factory”, *Submitted to the 5th International Conference on Quality Software (QSIC)*, 2005.

⁴ <http://www.componentsource.com>