Software Components Evaluation: an Overview

Miguel Goulão, Fernando Brito e Abreu

Departamento de Informática

Faculdade de Ciências e Tecnologia / Universidade Nova de Lisboa

Quinta da Torre, 2829-516 Caparica, Portugal
{miguel.goulao|fba}@di.fct.unl.pt

Abstract

Objective: To contribute with an overview on the current state of the art concerning metrics-based quality evaluation of software components and component-based assemblies.
Method: Comparison of several approaches available in the literature, in terms of their scope, intent, definition technique and maturity.
Results: Common shortcomings of current approaches, such as ambiguity in definition, lack of adequacy of the specifying formalisms and insufficient validation of current quality models and metrics for software components.
Conclusions: Quality evaluation of components and component-based infrastructures presents new challenges to the Experimental Software Engineering community which are not conveniently dealt with by current approaches.

Keywords: Component-Based Software Engineering; Component Evaluation; Software Metrics; Software Quality.

1 INTRODUCTION

Component-based development (CBD) is playing an increasing role in the software industry [Bass et al. 2001; Williams 2000]. There is an economic push to such growth: the claim is that CBD allows the reduction of cost and time to market, while increasing software quality, through reuse [Szyperski et al. 2002]. The rationale is that cost savings can be obtained through economy of scale, while improved quality results from the reuse of such components in different environments and applications. Recently, a component broker conducted a case study with the cooperation of component producers [Brooke 2002]. Its goal was to estimate the return on investment of commercial-of-the-shelf components (COTS). The referred case study reports that the costs of acquiring such components are about 1/50 of the ones of developing their required functionalities from scratch.

The Software Engineering Institute (SEI) defines a component as *"an opaque implementation of functionality subject to third party composition and conformant to a component model"* [Bachman et al. 2000]. With an increasing percentage of component-based architectures relying on black-box software components, the quality of such architectures depends, to a large extent, on the quality of those components and on the interactions among them [Simão et al. 2003]. Therefore, components evaluation should be integrated in CBD [Brownsword et al. 2000].

A component assembler takes application requirements, searches component repositories for selecting appropriate components and assembles them, by providing the required glue [Szyperski et al. 2002]. His focus of attention is component composition rather than component construction. For the component assembler, it is important to assess the complexity of alternative component assemblies, integrating components that may be acquired from different providers. Deciding whether to reuse components or to develop the corresponding functionality from scratch is also part of his tasks.

In this context, it would be helpful for a component assembler to have an integrated view of existing techniques that may assist him in this task. As pointed out in [Kitchenham et al. 2004], empirical software engineering research, in general, tends to be fragmented and not properly integrated. This leads to the absence of a culture of replication of experiments and of systematic

reviews of the existing approaches, like, for instance, is common practice for medical researchers [Pai et al. 2004]. Therefore, although approaches to component reusability assessment have been proposed, to the best of our knowledge, there is a lack of comparative reviews of such proposals. Furthermore, there are no standard protocols[1] to conduct individual experiments. Our contribution in this paper is twofold: we provide a comparative study on component reusability evaluation proposals and we outline an approach to support the replication of experiments to assess such proposals.

This paper is organized as follows: in section 2 we briefly discuss why software component characteristics present new challenges for the software engineering community; in section 3, we present a framework for our review on the state of the art of metrics-based assessment for CBD; in section 4 we discuss different metrics-based approaches to component evaluation, both in isolation and in assemblies; our view on how the state of the art may be improved is presented on section 5; finally, conclusions are presented in section 6.

2 THE NEED FOR CBSE SPECIFIC EVALUATION TECHNIQUES

Building upon SEI's definition, we can contrast evaluation of CBD with that of object-oriented or structured development. The first major difference relates to the opaqueness of components. While several metrics-based approaches for evaluation of software complexity (e.g: McCabe metrics) rely on access to the source code, similar approaches for CBD should depend only on the information publicly available on black-box components. Indeed, the component's source code is often not available to component assemblers. Moreover, there is a problem of scope. The component assembler is not concerned with the internal complexity of a component, but rather with the complexity involved in reusing it. Internal code metrics for analysing the components are not useful, from his point of view. Instead, complexity analysis on the interface of a component, the contracts associated with it and the adaptability of the component to different contexts should be assessed.

There is an effort to extend the ISO9126 [ISO9126 1995], to fit the needs of CBD [Simão et al. 2003; Bertoa et al. 2002]. Such a model is required for quality evaluation, whether this evaluation is of a qualitative or quantitative nature. A typical example of qualitative evaluation is an expert's opinion on the component artefact. Qualitative evaluation is subjective, posing problems in results comparison and generalization. Besides, experts may not be available at all.

The quantitative approach to evaluation provides, in our opinion, a more pragmatic way of dealing with this problem. It consists on defining, collecting and analyzing objective quantitative metrics that can be combined into a quality model to replace the expert's opinion in an automated fashion. The goal is to provide heuristics-based help as guidance to practitioners in the component selection process.

3 A FRAMEWORK FOR CHARACTERIZATION OF PROPOSALS

It is useful to have a common framework, upon which we can characterize the reviewed work. Framework-based analysis fosters a more systematic approach to proposals assessment than the one usually achieved through a more traditional narrative review. Evidence collected in the realm of medical sciences show that narrative reviews tend to lead to more informal and subjective methods to collect and interpret the studies and even to selective citation of literature to reinforce preconceived notions [Pai et al. 2004]. In contrast, having a framework for characterizing proposals fosters a more objective analysis, partially mitigating the shortcomings

[1] In the context of experimentation, a protocol clearly specifies the steps to be followed while conducting an experiment, from the experiment setup to data analysis. Following standard protocols increases the comparability of individual studies, as it fosters homogeneity in the data collection process.

of narrative reviews. Therefore, we propose here a framework upon which we will base our review.

- **Scope** – granularity level of the proposal
- **Intent** – main objectives of the proposal
- **Technique** – how the metrics are defined and validated
- **Critique** – a qualitative assessment of the noticeable features of the proposal
- **Maturity** – the maturity level of the proposal

The different proposals will be presented using the previous structure. The first four items of this structure aim to provide a very brief overview of the proposals, while the last aims to characterize each proposal according to its maturity level. To assess the maturity of the proposals, we start by identifying a set of rating scales concerning different aspects of metrics-based quality evaluation. For each of those rating scales, we then identify several levels of maturity that will aid us in the graphical depiction of proposals maturity. Table 1 presents a condensed view of our maturity comparison framework.

Maturity level	Quality Model (**QM**)	Mapping Quality (**MQ**)	Metrics definition (**MD**)	Level of Validation (**LV**)
0	N/A	N/A	N/A	N/A
1	Ad-hoc	Ad-hoc	Wish list	Anecdotal
2	Structured	Rationale	Informal	Small experiment
3	Uncorrelated	Goal-driven	Semi-formal	Industrial experiment
4	Validated	Validated	Formal	Independent

Table 1 - A metrics proposal maturity comparison framework

The maturity level is of an ordinal nature, ranging from 0, where no contribution is included in the proposal (N/A in all rating scales), to 4, where the proposal has reached a high maturity level. It should be noted that several of the proposals result from research work still in progress. In this sense, current maturity is orthogonal to the potential interest of the proposal. Rather than evaluating the merit of the proposal, it helps understanding how far the authors went with the presented work. For presentation purposes, we will use the following maturity mask, where level is replaced by the appropriate value for each proposal:

QM[level]; MQ[level]; MD[level]; LV[level]

The **Quality Model (QM)** represents the extent to which the metrics proposals fit into a quality model. For the Quality Model, the identified categories, by increasing level of maturity, represent:

1. **Ad-hoc** – A set of quality characteristics are identified.
2. **Structured** – Quality characteristics are organized, typically in a hierarchy.
3. **Uncorrelated** – Quality characteristics are shown to be independent, to avoid assessing the same quality aspect repeatedly.
4. **Validated** – The quality model is conveniently validated through experiments.

The **Mapping Quality (MQ)** represents the level of integration between the model and the metrics which are chosen to assess quality based on that model. The represented categories are:

1. **Ad-hoc** – Metrics are mapped to quality attributes in an ad-hoc fashion.
2. **Rationale** – A discussion on the rationale of the mapping is provided.
3. **Goal-driven** – Metrics are defined to answer specific evaluation needs, following an approach such as the Goal Question Metric [Basili et al. 1994].
4. **Validated** – Building on the previous level, metrics are shown to effectively fulfill the specific evaluation needs raised by the quality model.

Concerning **Metrics Definition (MD)**, we use the following categories:

1. **Wish list** – The authors informally identify the need for a certain kind of metrics, without actually proposing any metrics.
2. **Informal** – A natural language description of the metrics is provided by the authors.
3. **Semi-formal** – Some degree of formalism is used in the metrics definitions. Typically, the metrics themselves are defined through mathematical expressions, but the underlying concepts being measured are only informally specified.
4. **Formal** – A formal definition of the metrics and underlying concepts is provided.

Finally, the **Level of Validation (LV)** is classified according to the following categories:

1. **Anecdotal** – Anecdotal examples are provided to motivate the usefulness of the proposed metric. Sometimes, they are complemented with some descriptive statistics.
2. **Small experiment** – An experiment is carried out to assess the metrics, with some statistical approach to analyze the collected data, but the sample of analyzed artifacts does not allow generalizing conclusions.
3. **Industrial experiment** – An experiment with a significant sample of artifacts is carried out, with real-world artifacts and adequate statistical analysis.
4. **Independently validated** – Experiments made by independent research teams confirm the conclusions and claims made by the metrics proponents.

4 STRUCTURAL EVALUATION FOR CBSE

4.1 Approaches to the evaluation of individual components

In this section, we briefly discuss several approaches found in the literature, concerning the evaluation of individual components.

Bertoa's quality model and metrics [Bertoa et al. 2002; Bertoa et al. 2004]

Scope	COTS software
Intent	To introduce a quality model as an adaptation of the ISO9126 for component-based development [Bertoa et al. 2002]. The adaptation of the ISO quality model consists on assuming that the software will include black-box components and change the quality model accordingly, so that any assessment of reused software takes into account this restriction. A set of metrics to assess the attributes of that quality model is also proposed. Its rationale is that the metrics collection has to be defined considering the information made available by component brokers. While the first attempt at metrics definition covers transversally the quality model, more recent work by the same authors focuses on the usability of components, as

perceived by component assemblers [Bertoa et al. 2004].

Technique Although some of the metrics definitions include mathematical formulae, most definition are informal.

Critique By using the information made available by vendors, there are limitations concerning the ability to automate metrics collection, due to the noticeable lack of standards in data presentation by COTS producers and brokers. To overcome this problem, a UML model for the classification of COTS usability is proposed, but populating that model in an automated fashion remains an open challenge.

Maturity QM [Structured]; MQ[Rationale]; MD[Informal]; LV[N/A].

Gill's quality attributes [Gill et al. 2003]

Scope Black-box components

Intent To propose a set of guidelines on how to select metrics for black-box components.

Technique No actual metrics are defined. Instead, the authors informally present a set of quality attributes that should be evaluated through metrics.

Critique The proposal includes an interesting discussion on the focus shift caused by the specificity of black-box components evaluation, as opposed to evaluation of OO design, or structured software and provides an interesting roadmap for research in metrics-based component evaluation.

Maturity QM[Ad-hoc]; MQ[Rationale]; MD[Wish list]; LV[N/A].

Dumke's metrics for reusability of JavaBeans [Dumke et al. 2000]

Scope White-box Java Beans

Intent To present a metrics set for reusability of JavaBeans.

Technique Informal definition of metrics, relying on access to the source code. The metrics in this set are adapted from other contexts, such as OO design (e.g. percentage of public methods) and structured programming (e.g. maximal McCabe complexity number, for a method in the JavaBean class).

Critique The white-box view of components renders this approach inadequate for evaluation by independent component assemblers. The internal complexity of a component method should not be relevant for the understandability of its interface and the component's reusability.

Maturity QM[N/A]; MQ[Ad-hoc]; MD[Informal]; LV[Anecdotal].

Boxall's interface textual complexity metrics [Boxall et al. 2004]

Scope Interfaces of components developed with C, C++, Java or Eiffel.

Intent To define a set of metrics to assess interface complexity, measuring aspects of components' interfaces, such as the interface size, number of distinct arguments in operations, level of repetition of such arguments, the commonality in identifiers, identifier's length and the density of reference arguments.

Technique Metrics are defined through a set of mathematical expressions, but the elements of such expressions are informally described.

Critique The level of detail in the analysis of arguments in the interface is richer than in other approaches, in what concerns the relevance of naming conventions for component interfaces' understandability. However, this approach does not address other potentially interesting aspects in the interface, such as arguments' complexity.

Maturity QM[Informal]; MQ[Rationale]; MD[Semi-formal]; LV[Small experiment].

Washizaki's reusability metrics for black-box components [Washizaki et al. 2003]

Scope JavaBeans Interfaces.

Intent To propose a metrics set for assessing the reusability of JavaBeans. The metrics set is defined in the scope of a quality model for black-box component reusability, considering understandability, adaptability and portability as relevant sub-characteristics. More refined criteria are then defined for each of these sub-characteristics, as well as metrics to assess JavaBeans in light of such criteria.

Technique Metrics are defined as ratios of the effective use of a given sort of interface feature (e.g. BeanInfo class, readable properties, writable properties, methods with parameters and methods with no return value) when compared to its potential use. One such ratio is the number of readable properties over the total number of properties in the component.

Critique It can be argued that the analysis of the interface complexity is over-simplistic in at least two ways: (i) the inherent complexity of using simple arguments vs. complex ones, and (ii) the inherent complexity of using repeated argument types vs. different ones. In both cases no distinction is made. Intuitively, increased complexity and variety of argument types would decrease the understandability of the component's interface.

Washizaki's metrics set was validated with a case study where the reusability of over 120 components was assessed, both with this metrics set and by a panel of experts. Results show a high correlation between both assessments, indicating that the metrics defined in this set can indeed be used to assess component's reusability. However, an independent experiment showed the metrics to be unreliable for components with a small number of features on their interface [Goulão et al. 2004b]. Further independent analysis is still required.

Maturity QM[Structured]; MQ[Rationale]; MD[Semi-formal]; LV[Industrial experiment].

Gill's interface complexity metrics [Gill et al. 2004]

Scope Black-box components' interface

Intent Besides the complexity aspects of interfaces' signature, this proposal also considers constraints upon those interfaces, as well as their packaging, to account for different configurations that the interface may present, depending on the context of use.

Technique The overall complexity is defined as the weighted sum of the complexities related to signature, constraints and packaging of the interfaces. For each of these aspects of interface complexity, a definition is also proposed, again using weighted sums of features (e.g. events and operations count, for signature's complexity).

Critique Although Gill's proposal has the merit of including constraints and packaging complexities on the assessment, it still lacks any sort of empirical assessment. This hampers the ability of the authors to assign values to the coefficients on their definitions, and, more significantly, our ability to assess the extent to which this approach helps common practitioners to choose among alternative components.

Maturity QM[N/A]; MQ[N/A]; MD[Informal]; LV[N/A].

4.2 Approaches to the evaluation of component assemblies

The approaches described in the previous section are mostly targeted at the assessment of components in isolation. They rely on the assumption that the quality of software components influences in some way the quality of the assembled system. The apparent conclusion of this would be that a component assembler should always try to choose the best components in order to optimize the quality of the assembled system. This may reveal to be naïf, since we should also consider the context under which the component will operate. Determining how well a component integrates with other components in an assembly may lead to an evaluation that is more worthy to the component assembler than the one made in isolation [Wallnau et al. 2002]. This change of scope to the component assembly allows the component assembler to focus on the quality for his target product: the component assembly.

Sedigh-Ali's quality characteristics [Sedigh-Ali et al. 2001]

Scope COTS

Intent To discuss the requirements for metrics for CB-architectures, based on relevant quality aspects. The authors also present a taxonomy on the categories of costs related to software quality, with cost drivers such as quality improvement, low quality prevention, software failures and external costs related to those failures.

Technique High level discussion, rather than a concrete proposal.

Critique The main contribution of this paper is an interesting discussion on requirements for metrics for CB architectures, measured at a system level, including insights on how to choose relevant metrics. However, this is an exploratory work based on expert opinions alone, rather than on some sort of quantitative evidence to back up the assumptions made.

Maturity QM[Ad-hoc]; MQ[Ad-hoc]; MD[Wish list]; LV[N/A].

Seker's coupling and cohesion for CBD [Seker 2004]

Scope Black-box components and component assemblies

Intent To define coupling and cohesion metrics for CBD.

Technique The metrics are defined using an information theory based approach where components and component infrastructures are represented as graphs.

Critique This approach adapts the well-known concepts of coupling and cohesion to the scope of CBD. Except for the nodes in the graph being black-box components rather than classes, the proposal is similar to coupling and cohesion for OO design.

Maturity QM[N/A]; MQ[N/A]; MD[Semi-formal]; LV[N/A].

Hoek's service utilization metrics [Hoek et al. 2003]

Scope	Software product lines
Intent	To propose a metrics set that allows assessing software product lines based on service utilization. The rationale for their need is that they imply a degree of optionality among the components that get used in a given configuration. While some will be part of all configurations of that product line, others are optional. Structural variability is also an issue, as one has to choose among a range of alternative configurations. Product lines are also typically hierarchical, composed of a set of components, each of which with its own internal structure. These constraints violate the assumption of most structural metrics that the system structure under evaluation is: (i) single - no optionality considered, snapshots of the system are usually evaluated); (ii) fixed - no structural variability, the system structure is assumed to be kept constant throughout the evaluation; and (iii) flat - the implications of the hierarchical decomposition of the system are not considered in the metrics definition).
Technique	The metrics are based upon existing metrics and defined around the concept of service utilization (the rate of usage of provided and required services of a component). For individual components, metrics are simply ratios of used services (both for required and provided ones), whereas for fixed, flat component architectures (assemblies) these ratios are obtained using the sum of used services against the total of available services.
Critique	Of all the proposals presented in this overview, Hoek's is the one that best fits the notions of architectural components and assemblies' evaluation rather than individual components' evaluation.
Maturity	QM[N/A]; MQ[N/A]; MD[Semi-formal]; LV[Anecdotal].

4.3 Abstracting some lessons learned

Figure 1 represents an overview of the maturity levels of each of the proposals described on previous sections. In this chart, from left to right, we present each proposal; from the front to the back we present each of the analysed rating scales; on the vertical axis we have the maturity level, as defined in Table 1. The overall low level of maturity throughout the several rating scales contained in our proposal suggests that research in the area of software components evaluation with the usage of software metrics is still on a very early stage. For instance, it is worth noticing that only Washisaki's proposal was validated with an industry-level experiment, while most proposals were not validated at all.

Across these proposals, we have identified a number of common problems with metrics definitions that help explaining this state of affairs. These problems are not specific to CBSE. Indeed we have observed similar problems with metrics proposals for OO design [Abreu 2001b]. Among them, we would like to highlight the following:

- **lack of context** – occurs when metrics definition is not performed within the scope of the requirements of a particular quality model.
- **ambiguity in definition** – occurs when metrics are defined using informal definitions; this may cause misleading interpretations and, therefore, different implementations of metrics collection tools may produce different results on the same software artefacts, if different assumptions are made on what is considered as relevant to the metrics computation.

- **inadequate specifying formalism** – occurs when metrics are defined through some formalism that requires a strong mathematical background, often not held by practitioners.
- **insufficient validation** – occurs when independent cross validation is not performed, mainly due to difficulties in experiment replication; independent metrics validation (not performed by their authors) is fundamental for their proof of usefulness before widespread acceptance is sought.

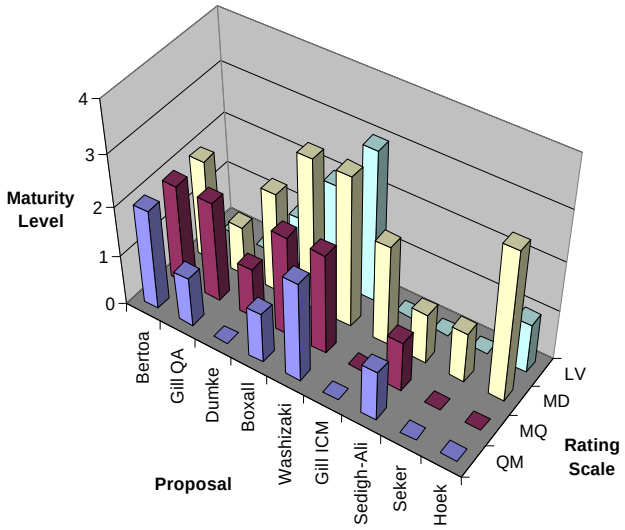

Figure 1 – *Overall proposal maturity assessment*

5 MITIGATING SOME OF THE IDENTIFIED PROBLEMS

Automated metrics extraction is fundamental to foster independent validation efforts. Unfortunately, most of the proposed metrics are only tested by their authors, using proprietary tool support, therefore limiting their portability. This limits knowledge sharing, both in the research and practitioner's communities, hampering results comparison. We have proposed elsewhere an approach to mitigate this problem [Goulão et al. 2004b], which relies on the usage of a metamodel to formally define the concepts we aim to measure, and OCL expressions to define metrics over that metamodel. It can be summarized as follows:

- producing formal and executable metrics specifications, using OCL 2.0 [OMG 2003b] which is a part of the UML 2.0 standard;
- tying the formal metrics specification to a metamodel, such as the UML 2.0 one [OMG 2003a; OMG 2003c], thus ensuring that there are no definition and computing ambiguities;
- packaging metrics specifications in a format that can shared with other researchers and practitioners, either for validation purposes, or for usage in their daily activities.

Our proposal for metrics definition and collection combines formality, understandability and collection efficiency due to the usage of OCL. Furthermore, it ensures their portability among OCL-enabled CASE tools. As OCL is part of the new UML standard, an increasing number of UML CASE tools are supporting it. To collect the metrics, we execute their corresponding OCL definition upon the referred metamodel, instantiated with meta-objects representing the component assemblies to be analyzed.

Further details about this technique and case studies that illustrate its applicability have been published in the recent years. In [Abreu 2001b; Abreu 2001a], the idea of using OCL for defining software metrics to evaluate object-oriented desing is proposed. [Baroni et al. 2002] presents FLAME, a library of OCL functions to aid in the definition and extraction of software metrics, based on the UML metamodel. [Goulão et al. 2004a] moves to the evaluation of software components and discusses the formalization of Washizaki's metrics set [Washizaki et al. 2003], using the new UML 2.0 metamodel. This formalization uses the new abstractions for software components provided by UML 2.0. [Goulão et al. 2004b] presents a case study to assess Washizaki's metrics set in a quantitative way.

6 CONCLUSIONS AND FURTHER WORK

With the increasing demand of the software industry to include third party reusable components in the software development process, component assemblers need effective ways of selecting adequate components. Comparative reviews of existing approaches to software component evaluation are required to aid component assemblers to identify the evaluation approaches better suited to support their activity. As far as we know, this is the first attempt to provide such a review in a systematic way.

In this paper, we contribute with a common framework for the characterization of component assessment classification proposals. The framework includes the proposal's scope, intent and used technique, a critical appreciation of the most noticeable features of the proposal and an assessment of its maturity level, regarding the underlying quality model, its mapping to metrics, the metrics definition and the achieved level of validation.

Common problems on current approaches to CBD evaluation are identified. Overall, there is a lack of maturity in existing proposals, which is likely due to the relative novelty of black-box software components evaluation as a research topic. For instance, determining the relevant quality attributes which should be assessed is still an open issue. Ambiguity in definition of quality models and metrics, lack of adequacy of specifying formalisms and insufficient validation of proposals are among the most common shortcomings in the analysed proposals. We briefly outline our approach to mitigate these problems.

Our analysis focuses on metrics-based evaluation of structural properties of components and component assemblies. This is only part of the "evaluation toolset" that should be available to component assemblers. Other orthogonal evaluation techniques, such as the evaluation of non-functional properties and, obviously, the assessment of composability of candidate components are essential to component selection, but are out of the scope of the review performed in this paper.

We plan to expand and sophisticate further this review. On the one hand, the review can be updated with new proposals that are being published on the literature. On the other, the review framework itself can be improved in several ways. Several categories of our framework can be further refined, to provide more details concerning the metrics definition and validation techniques and so forth.

Again, we can find inspiration on the medical sciences systematic reviews for other improvements: The method for gathering the proposals to be reviewed can and indeed should be presented using clear and reproducible search and eligibility criteria. This is currently not easily implementable in the realm of experimental software engineering, due to the diversity and scattering of information, but is common practice for clinical research, where evidence-based healthcare databases are maintained (http://www.cochrane.org/).

Research networks such as the ESERNET (Experimental Software Engineering Network - http://www.esernet.org/) plan to coordinate efforts to mitigate this problem. Empirical software engineering case studies are often conducted in an ad-hoc fashion. Striving for well-defined

protocols in metrics collection experiments would significantly improve the comparability between different proposals. In the long run, we believe it is desirable that this sort of systematic reviews can be made upon case studies following well-defined experimental protocols, so that meta-analysis of the experimental data collected by independent research teams is more feasible. As members of the ESERNET, we expect to contribute to this objective, as well.

REFERENCES

Abreu, F. B. Engenharia de Software Orientado a Objectos: uma Aproximação Quantitativa, IST (2001a).

Abreu, F. B. Using OCL to formalize object oriented metrics definitions. Software Engineering Group, INESC (2001b).

Bachman, F., Bass, L., Buhman, C., Cornella-Dorda, S., Long, F., Robert, J., Seacord, R. and Wallnau, K. Volume II: Technical Concepts of Component-Based Software Engineering, Software Engineering Institute (2000).

Baroni, A. L. and Abreu, F. B. *Formalizing Object-Oriented Design Metrics upon the UML Meta-Model*. Brazilian Symposium on Software Engineering, Gramado - RS, Brazil. (2002)

Basili, V. R., Caldiera, G. and Rombach, D. H. Goal Question Metric Paradigm. *Encyclopedia of Software Engineering*. J. J. Marciniak, John Wiley & Sons. 1.(1994). 469-476.

Bass, L., Buhman, C., Comella-Dorda, S., Long, F., Robert, J., Seacord, R. and Wallnau, K. Volume I: Market Assessment of Component-Based Software Engineering, Software Engineering Institute (2001).

Bertoa, M. and Vallecillo, A. *Quality Attributes for COTS Components*. 6th International Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE'2002), Málaga, Spain. (2002)

Bertoa, M. and Vallecillo, A. *Usability metrics for software components*. QAOOSE 2004, Oslo. (2004)

Boxall, M. A. S. and Araban, S. *Interface Metrics for Reusability Analysis of Components*. Australian Software Engineering Conference (ASWEC'2004), Melbourne, Australia. (2004)

Brooke, C. The Return on Investment on Commercial off-theshelf (COTS) Software Components - Preliminary Study Results. Kennesay, GA, USA - Reading, UK, Component Source (2002).

Brownsword, L., Oberndorf, T. and Sledge, C. A. "Developing New Processes for COTS-Based Systems." *IEEE Software* (2000). 48-55.

Dumke, R. and Schmietendorf, A. "Possibilities of the Description and Evaluation of Software Components." *Metrics News* 5(1) (2000).

Gill, N. S. and Grover, P. S. "Component-Based Measurement: Few Useful Guidelines." *ACM SIGSOFT Software Engineering Notes* 28(6) (2003).

Gill, N. S. and Grover, P. S. "Few Important Considerations for Deriving Interface Complexity Metric for Component-Based Software." *Software Engineering Notes* 29(2) (2004).

Goulão, M. and Abreu, F. B. *Formalizing Metrics for COTS*. MPEC'2004, Edimburgh. (2004a)

Goulão, M. and Abreu, F. B. *Independent Validation of a Component Metrics Suite*. IX Jornadas de Ingeniería del Software y Bases de Datos, Málaga, Spain. (2004b)

Hoek, A. v. d., Dincel, E. and Medvidovic, N. *Using Service Utilization Metrics to Assess and Improve Product Line Architectures*. Ninth International Software Metrics Symposium (Metrics'03), Sydney, Australia, IEEE Computer Society Press. (2003)

ISO9126. Information Technology - Software Product Evaluation - Software Quality Characteristics and Metrics. Geneva, Switzerland, International Organization for Standardization (1995).

Kitchenham, B., Dyba, T. and Jorgensen, M. *Evidence-based Software Engineering*. 26th International Conference on Software Engineering, Edinburgh, Scotland, IEEE Computer Society Press. (2004)

OMG. UML 2.0 Infrastructure Final Adopted Specification, Object Management Group, Inc. (2003a).

OMG. Unified Modeling Language: OCL (version 2.0), Object Management Group Inc. (2003b).

OMG. Unified Modeling Language: Superstructure - Version 2.0 - Final Adopted Specification, Object Management Group Inc. (2003c).

Pai, M., McCulloch, M., Gorman, J. D., Pai, N., Enanoria, W., Kennedy, G., Tharian, P. and Colford, J. M., Jr. "Systematic reviews and meta-analyses: An illustrated step-by-step guide." *National Medical Journal of India* 17(2) (2004). 86-95.

Sedigh-Ali, S., Ghafoor, A. and Paul, R. A. "Software Engineering Metrics for COTS-Based Systems." *IEEE Computer* (2001).

Seker, R. *Assessment of Coupling and Cohesion for Component-Based Software by Using Shannon Languages*. South African Institute of Computer Scientists and Information Technologists, Stellenbosch, Western Cape, South Africa. (2004)

Simão, R. P. S. and Belchior, A. D. Quality Characteristics for Software Components: Hierarchy and Quality Guides. *Component-Based Software Quality: Methods and Techniques*. A. Cechich, M. Piattini and A. Vallecillo, Springer(2003). 184-206.

Szyperski, C., Gruntz, D. and Murer, S. *Component Software: Beyond Object-Oriented Programming*. New York, ACM Press - Addison Wesley (2002).

Wallnau, K. and Stafford, J. A. Dispelling the Myth of Component Evaluation. *Building Reliable Component-Based Software Systems*. I. Crnkovic and Larsson. Boston, London, Artech House(2002). 157-177.

Washizaki, H., Yamamoto, H. and Fukazawa, Y. *A Metrics Suite for Measuring Reusability of Software Components*. Metrics'2003. (2003)

Williams, J. D. "Raising Components." *Application Development Trends* 7(9) (2000). 27-32.