Usability Indicators for Software Components

Manuel F. Bertoa and Antonio Vallecillo

Dpto. Lenguajes y Ciencias de la Computación. Universidad de Málaga.
{bertoa,av}@lcc.uma.es

Abstract. One of the most critical processes of Component-based Software Development (CBSD) is the selection of the set of components (either from in-house or from external repositories) that fulfil the appropriate architectural and user-defined requirements. This process opens the need to count with objective methods that help developers evaluate the components. This paper presents a set of measures and indicators to assess one quality characteristic, the Usability, of great importance to any software product, and describes the method followed to obtain and validate them.

Introduction

Component-based software development (CBSD) has become an important alternative for building software applications, and in particular for distributed systems. CBSD tries to improve the flexibility, re-usability and maintainability of applications, helping develop complex and distributed applications deployed on a wide range of platforms, by plugging commercial off the-shelf (COTS) components, rather than building these applications from scratch. The goal is to reduce the development costs and efforts, while improving the quality of the final product due to the (re)use of software components already tested and validated.

Initially, there was a complete absence of metrics that could help evaluate software component quality attributes objectively. The international standards in charge of defining the quality aspects of software products (e.g. ISO/IEC 14598 [1] and ISO/IEC 9126 [2] series) are currently under revision. The SQuaRE project [3] has been created specifically to make them converge, trying to eliminate the gaps, conflicts, and ambiguities that they currently present. A drawback of the existing international standards is that they provide very general quality models and guidelines, but are very difficult to apply to specific domains such as CBSD and COTS. In this sense, emergent proposals in this area try to define quality models for COTS components and for component-based systems [4,5,6,7,8,9].

In recent works, we tried to adapt the general ISO/IEC 9126 Quality Model to the realm of software components, for which a set of measures was proposed [4]. Then, we looked at the information provided by software component vendors, analyzing the information they currently provide about the components they sell or license, with the purpose of determining how many of these measures were in fact computable. We found out that the information provided by vendors is normally scarce and mostly insufficient for any quality analysis [10].

The assessment of the quality of a software component is in general a very broad and ambitious goal. For instance, ISO/IEC 9126 Quality Model defines the quality of a software product in terms of six major characteristics (Functionality, Reliability, Usability, Efficiency, Maintainability and Portability), which are further refined into 27 sub-characteristics. Here we will just concentrate on one of these quality characteristic, the Usability, because of its importance for CBSD. Usability is inherent to software quality because it expresses the relationship between the software and its application domain.

In this paper we thus present a set of measures to assess the Usability of software components. Furthermore, we describe the process followed to obtain and validate them. Such a process can be (re-)used for defining and validating measures for other quality characteristics. This paper extends our previous work [12] by proposing a set of indicators, based on the base and derived measures defined there.

The paper is organized as follows. After this introduction, Section 2 identifies the factors that influence component Usability and defines a set of measures. The process followed for validating the proposed measures is described in Section 3. Then, Section 4 describes a set of indicators, based on the derived and base measures. The paper finishes by drawing some conclusions and outlining some future research activities.

Defining Usability Measures

When we surveyed the literature looking for software component measures, we discovered that the relationship between the measures and the quality characteristics they tried to assess was ill-defined. This was a common problem of most of the proposals and International Standards that defined software component measures.

Our aim here is to solve that problem, trying to properly define a set of measures based on the attributes they evaluate, and determine how these measures assess the quality characteristics of a software component, according to a given quality model.

We will also impose some requirements on the defined measures to make them feasible and practical in industrial environments: first, they need to be *objective* and *reproducible*; second, they need to be *easy to compute*--and even better if they are amenable to automatization; and finally, they should allow us to identify a minimum set of measures that provide the maximum amount of information about the measurable concepts we are trying to evaluate (i.e., optimize the number of representative measures).

To define the Usability measures we need to define in the first place an *information need*, which in this case is "to evaluate the Usability of a set of software components that are candidates to be integrated in a software system, in order to select the best among them".

At least three measurable concepts are closely related to software component Usability: the *Quality of the Documentation*, the *Complexity of the Problem* and the *Complexity of the Solution* (or *Design*). As we are comparing software components that provide similar functionality to resolve the same kind of problem, we will assume that all of them share the same *Complexity of the Problem* and, accordingly, there is

no need to propose measures to evaluate this measurable concept. Hence, we will focus on the other two. These measurable concepts and their related attributes are presented in Table 1. The way in which these measurable concepts influence the Usability of a component and its related quality sub-characteristics will be addressed later in sections 3 and 4.

Entity	Information Need	Measurable Concept		Attribute
Software Component	Evaluate the Usability	Quality of Documentation	Quality of Manuals	Contents of manuals
				Size of Manuals
				Effectiveness of Manuals
			Quality of Demos	Contents of Demos
			Quality of Marketing Info	Contents of Marketing Info
		Complexity of the Design		Design Legibility
				Interfaces Understandability
				Learning facility
				API complexity
				Customisability

Table 1. Measurable Concepts and Attributes for Usability

Since each attribute may have one or more measures (base, derived, or indicators) that evaluate it, we need to define these measures. The following tables show the measures proposed to measure the measurable concepts defined in Table 1. A summary of the measures defined for the *Quality of the Documentation* attributes and for those related to the *Complexity of Design* are shown in Table 2. We use the term "functional element" to refer interfaces, operations and configurable parameters as a whole.

There should also be a last column with the base measures on which derived measures are defined, but this column has been omitted for space restrictions. The majority of these base measures are evident (for instance, to measure the ratio of figures per pages two base measures are required: the number of figures and the number of pages in the manual). Also, note that many *ratio* measures repeat their denominator, which produces an unnecessary repetition of rows (this is for instance the case of the number of functional elements that is used in many derived measures). Other cases are not so evident, like "COMPLETENESS OF MANUALS", which measures the number of functional elements of the component that do not appear in the manual.

ATTRIBUTE	INDICATOR	DERIVED MEASURES
CONTENTS OF MANUALS	MANUALS COVERAGE	% OF FUNCTIONAL ELEMENTS DESCRIBED IN MANUALS
		% OF INTERFACES DESCRIBED IN MANUALS
		% OF METHODS DESCRIBED IN MANUALS
		% OF CONFIG. PARAM. DESCRIBED IN MANUALS
	MANUALS CONSISTENCY	*PROPORTION OF FUNCTIONAL ELEMENTS INCORRECTLY DESCRIBED IN THE MANUAL*
		DIFFERENCE BETWEEN THE COMPONENT VERSION AND THE MANUAL VERSION
	MANUALS LEGIBILITY	RATIO OF HTML FILES OF MANUALS PER FE
		RATIO OF FIGURES PER KILO-WORD
SIZE OF MANUALS	MANUALS SUITABILITY	RATIO OF WORDS PER FE
		RATIO OF WORDS PER INTERFACE
		RATIO OF WORDS PER METHOD
		RATIO OF WORDS PER CONFIGURABLE PARAMETER
EFFECTIVENESS OF MANUALS	EFFECTIVENESS RATIO	*PERCENTAGE OF FUNCTIONAL ELEMENTS CORRECTLY USED AFTER READING THE MANUAL*
CONTENTS OF DEMOS	DEMOS COVERAGE	*PERCENTAGE OF FUNCTIONAL ELEMENTS INCLUDED IN DEMOS*
	DEMOS CONSISTENCY	*DIFFERENCE BETWEEN DEMO VERSION AND COMPONENT VERSION*
CONTENTS OF MARKETING INFO	MARKETING INFO COVERAGE	*NUMBER OF MARKETING INFO ELEMENTS DESCRIBED*
	MARKETING INFO CONSISTENCY	*DIFFERENCE BETWEEN THE COMPONENT VERSION AND THE MARKETING INFO VERSION*
DESIGN LEGIBILITY	MEANINGFUL NAMES	PERCENTAGE OF FE WITH LONG NAMES
		AVERAGE LENGTH OF FE NAMES
INTERFACES UNDERSTANDABILITY	FUNCTIONAL ELEMENTS UNDERSTANDABILITY	*PERCENTAGE OF FE USED WITHOUT ERRORS*
LEARNING FACILITY	TIME TO USE	*AVERAGE TIME TO USE CORRECTLY THE COMPONENT*
	TIME TO EXPERTISE	*AVERAGE TIME TO MASTER THE COMPONENT FUNCTIONALITY*
API COMPLEXITY	DENSITY OF INTERFACES	RATIO OF INTERFACES PER REQUIRED INTERFACE
		RATIO OF RETURN VALUES PER METHOD
		RATIO OF METHOD ARGUMENTS PER METHOD
		PERCENTAGE OF METHODS WITHOUT ARGUMENTS
		RATIO OF METHODS PER INTERFACE
		RATIO OF CONSTRUCTORS PER CLASS
		RATIO OF CONSTRUCTORS PER METHOD
CUSTOMISABILITY	CUSTOMISABILITY RATIO	RATIO OF CONFIG. PARAM. PER INTERFACE
		RATIO OF CONFIG. PARAM. PER METHOD

Table 2. Usability Derived Measures for COTS components

Validating Usability Measures

The goal of the validation is to prove that a measure really provides useful information to assess a quality characteristic. In order to empirically validate our measures we conducted a set of experiments. They tried to provide us with some figures (i.e., numerical values) about the Understandability, Learnability and Operability of a set of software components:

DERIVED MEASURE	D_USAB	I_USAB	O_USAB	O_UND	O_LEAR	O_OPER
% of FE described in manuals	0.04	0.29	0.61	0.47	0.63	0.57
% of Interfaces described in manuals	0.30	0.18	0.32	0.28	0.00	0.05
% of Methods described in manuals	-0.21	-0.06	0.35	0.41	0.38	0.28
% of Config. Param. described in manuals	-0.03	0.37	0.20	-0.25	0.21	0.20
Component Version difference	0.44	0.50	0.42	0.58	0.69	0.67
Ratio of Manual HTML files per FE	0.43	0.69	**0.93**	0.78	0.83	0.78
Ratio of figures per word	-0.08	0.45	0.22	-029	0.18	0.19
Ratio of words per FE	0.32	0.59	0.85	0.68	**0.91**	**0.90**
Ratio of words per Interface	0.13	0.22	0.46	0.42	0.68	0.73
Ratio of words per Method	0.35	0.61	0.88	0.73	**0.91**	**0.90**
Ratio of words per Config. Param.	-0.05	0.46	0.56	0.10	0.63	0.63
Percentage of FE with long names	0.17	0.27	-0.10	-0.10	-0.05	-0.10
Ratio of Interfaces per Req Interface	-0.44	-0.03	-0.23	-0.68	-0.21	-0.20
Ratio of Return Values per Method	-0.71	-0.44	-0.44	**-0.82**	-0.53	-0.53
Ratio of Arguments per Method	-0.35	-0.64	-0.59	-0.50	-0.78	-0.78
Percentage of Methods without Arguments	-0.46	0.16	0.15	-0.38	0.20	0.22
Ratio of Config. Param.per Interface	-0.15	-0.32	-0.03	0.22	0.09	0.13
Ratio of Config. Param. per Method	-0.03	-0.14	0.14	0.43	0.15	0.12
Ratio of Methods per Interface	-0.06	-0.14	-0.02	0.00	0.16	0.27
Ratio of Constructors per Class	0.44	0.72	0.49	0.23	0.63	0.67
Ratio of Constructors per Method	0.11	0.22	0.10	0.01	0.02	-0.08

Table 3. Correlation Values (R) for derived Measures

Five experiments were conducted between June and December 2004. The idea was to simulate the selection process of a software component from a set of potential candidates. We selected 12 software components from one application domain, *Print and Preview*, because it was neither too simple nor too complex, and because there were enough candidate components to constitute a representative sample. All candidate components were COTS products available from commercial vendors, advertised in component repositories such as ComponentSource. They used different component models and technologies, namely Java, ActiveX and .NET.

From all these experiments we obtained six variables for the sampled components: Direct Perceived Usability (D_Usab), Indirect Perceived Usability (I_Usab), Objective Understandability (O_Und), Objective Learnability (O_Learn), Objective Operability (O_Oper) and Objective Usability (O_Usab). The first ones correspond to the perceived Usability, measured directly and indirectly. The next three were

obtained from the results of the experiment's questionnaire. Finally, the Objective Usability is the average of the other three objective values.

BASE MEASURE	D_USAB	I_USAB	O_USAB	O_UND	O_LEAR	O_OPER
Number of words in manuals	-0.05	-0.08	0.08	0.12	0.38	0.44
Number of Manual (HTML) files	0.10	0.17	0.39	0.37	0.64	**0.68**
Number of Interfaces	0.12	-0.18	-0.39	-0.13	-0.08	-0.08
Number of Methods	-0.01	-0.15	-0.12	0.01	0.23	0.29
Number of Config. Parameters	-0.12	-0.29	-0.13	0.11	0.21	0.26
Number of Constructors	0.21	-0.04	-0.28	-0.07	0.03	0.05
Number of Method Arguments	-0.03	-0.22	-0.19	-0.02	-0.15	0.22
Number of Return Values	-0.09	-0.18	-0.14	-0.06	0.20	0.27
Number of Functional Elements	-0.03	-0.19	-0.13	0.03	0.22	0.28
Number of Interfaces in manuals	0.44	0.01	0.07	0.29	0.06	0.13
Number of Methods in manuals	-0.10	-0.22	-0.11	0.03	0.24	0.29
Number of Config. Param. in manuals	0.03	-0.06	0.03	0.12	0.35	0.41
Number of FE in manuals	-0.06	-0.18	-0.08	0.05	0.27	0.33
Number of figures in manuals	0.00	0.50	0.34	-0.01	0.34	0.35
Average length of FE names	0.29	0.10	-0.28	-0.07	-0.27	-0.30
Number of FE with long names	0.22	0.01	-0.30	-0.04	0.02	0.04
Number of Methods without Return Value	0.05	-0.13	-0.10	-0.03	0.25	0.32
Number of Methods without Arguments	-0.07	-0.14	-0.10	-0.03	0.25	0.32

Table 4. Correlation values (R) for Base Measures

A total of 68 users participated in the experiments evaluating the Usability of the sampled components. The number of users ranged between 9 and 18 depending on the experiment. They were Computer Science last-year students, researchers, and lecturers, all with enough programming skills and knowledge to build a system using simple commercial components. They were mostly from the University of Malaga, although one experiment was replicated in the University of Castilla-La Mancha.

Once we had quantified the information about the Usability of the sampled components and measured them using our defined measures, the next step was to check if the measures values really explained the (sub)characteristic values, which is given by the square of the linear correlation coefficient (R^2) between the measures and the corresponding quality characteristic. A strong correlation (IEEE [11] proposes $R^2 > 0.7$, but we looked for $R^2 > 0{,}95$) warrants using the measure as a substitute for the (sub)characteristic. Table 3 and Table 4 show, respectively, the correlations matrixes obtained for our base and derived measures.

One of the most interesting conclusions that can be drawn from the correlation tables is that no individual measure provides a really good explanation (i.e., with R^2 close to 1) of the Understandability, Learnability, or Operability of a component, or of its Usability as a whole. More precisely, only one measure on its own explained more than 82% of a quality characteristic (the ratio of words per FE, with $R = 0.92$, $R^2 = 0.83$). The rest of the measures explained even less than that. This is what moved us to look for new measures (indicators) defined as a combination of two or more base or derived measures. This is explained in the next section.

Defining Usability Indicators

At this point, we need to recall what we identified: the measures are usually assigned to just one quality subcharacteristic, although in theory they may evaluate more than one quality characteristic. In fact, we do not believe that there is a unique direct relationship between a measure and a quality sub-characteristic, but that there are different degrees of relation between every measure and every sub-characteristic.

Then, using the data obtained from the experiments and from our measures, we used linear regression analysis to look for combinations of measures that provided better explanations of the three quality sub-characteristics. Our findings were astonishing: all the Usability sub-characteristics could be accurately explained by linear combinations of two measures, obtaining values for R^2 around 0.97 .

These combinations are shown in Table 5 and, among other things, provide very interesting information about the existing links between the component attributes and the Quality Model, i.e., the connection between the Quality of Documentation and Complexity of Design, and the Understandability, Learnability and Operability of a software component:

- The Understandability strongly depends on both the ratio of HTML files per FE, and on the ratio of return values per method.
- The Learnability strongly depends on both the ratio of words per method and on the ratio of arguments per method.
- Finally, the Operability strongly depends on both the ratio of words per configurable parameter (or word per fields, in the case of Java components) and the ratio of return values per method.

Subcharacteristic	Depends on Measures		R^2	Relationship
Understandability (O_Und)	Ratio of HTML files of manuals per FE (Fil)	Ratio of Return Values per Method (RVpM)	0.970	O_Und= 0.200·Fil – 1,423·RVpM + 1.598
Learnability (O_Learn)	Ratio of Kilo-Words per Method (WpI)	Ratio of Arguments per Method (ApM)	0.973	O_Learn= 1.789·WpI – 2.090·ApM + 1.712
Operability (O_Oper)	Ratio of Kilo-Words per Configurable Parameter (WpCP)	Ratio of Return Values per Method (RVpM)	0.980	O_Oper= 0.804·WpCP –8.265·RVpM + 3.486

Table 5. Indicators for Usability subcharacteristics

Please notice that some of the equations in Table 5 do not include measures that were very influential on their own. This means that a combination of less representative measures may become more representative than the individual measure themselves, and than any other individual measure.

Now we have a new set of derived measures which properly explain the Usability sub-characteristics, we also need to define some criteria for knowing whether the component is acceptable or not with regard to each of these sub-characteristics. IEEE

[11] defines three categories for classifying software: (A)cceptable, (M)arginal, or (U)nacceptable.

We are not going to define a global Usability indicator for a software component, because the weight of each individual sub-characteristic heavily depends on the user and on the context of use. Instead, we propose a ternary tuple (U,L,O) for measuring the Usability of a component, where U, L, and O are the Understandability, Learnability, and Operability indicators described below. Thus, possible values of the tuple are (A,M,A) for a component with acceptable Understandability and Operability, and marginal Learnability; or (M, U, U) for a component with marginal Understandability, and unacceptable Learnability and Operability.

To define such indicators we have identified some critical values (thresholds) with discriminative power, i.e., that can determine whether a value of **O_Und**, **O_Learn** or **O_Oper** corresponds to a component with acceptable, marginal, or unacceptable Understandability, Learnability or Operability, respectively. Ideally, the values obtained for the sampled components should gather in two major groups (acceptable and unacceptable). Components outside these two groups will be considered as marginal.

Quoting IEEE [11], indicators should have *discriminative power*. "An indicator shall be able to discriminate between high-quality software components and low-quality software components. The set of indicator values associated with the former should be significantly higher (or lower) than those associated with the latter. This criterion assesses whether an indicator is capable of separating a set of high-quality software components from a set of low-quality components. This capability identifies critical values for indicators that shall be used to identify software components that have unacceptable quality. To perform this test, put the subcharacteristic and indicator data in the form of a contingency table and compute the chi-square statistic. This value shall exceed the chi-square statistic corresponding to a confidence level α."

	Acceptable	Marginal	Unacceptable
Understandability (**O_Und**)	1,0	0,9	0,7
Learnability (**O_Learn**)	1,0	0,6	0,2
Operability (**O_Oper**)	0,9	0,6	0,3

Table 6. Critical values for the three Usability sub-characteristics

Table 6 shows the critical values we have obtained for the aggregated functions that explain the Understandability, Learnability and Operability.

For instance, this means that we will say that a software component has an (A)cceptable Understandability if the value it obtains for **O_Und** is greater than 1.0. Its Understandability will be considered as (U)nacceptable if **O_Und** evaluates to a value which is less than 0.7. Finally, it will have a (M)arginal Understandability if the value it obtains for **O_Und** is between 0.7 and 1.0.

Figure 1 graphically shows these critical values in the case of the Learnability, whose thresholds are 0.2 and 1.0. Equating the aggregated function **O_Learn** to 1.0 and to 0.2 we obtain the two lines showed in the graphic. These lines divide the plane into three areas, each one corresponding to one classification (Acceptable, Unacceptable, and Marginal). The sampled components can therefore be classified

according to the values they obtain in the two aggregated measures (**WpM** and **ApM**) of **O_Learn**.

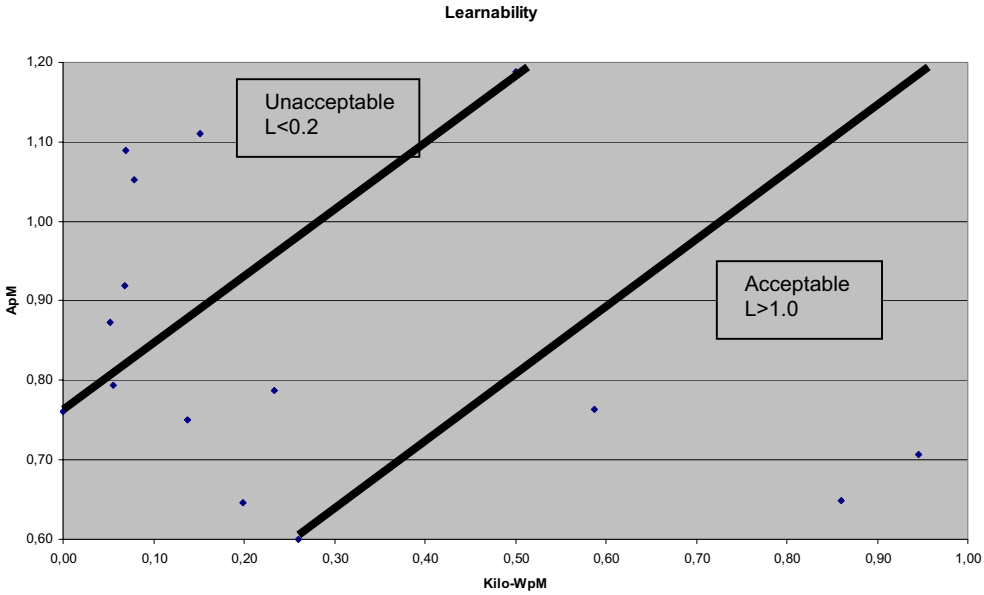

Figure 1. Measure Values of WpM and ApM used in Learnability Indicator

Discussion

This paper has presented a set of base and derived measures that can be used to evaluate the Usability of software components, and some indicators for classifying them according to IEEE categories.

One interesting result is that we have seen how the appropriate combinations of (base and derived) measures can evaluate better the Usability of a component than any individual measure. Basically, this is because quality characteristics do not depend on particular measurable concepts, but on combination of those.

In summary, we have shown that: (a) the Understandability seems to depend on the structure and organization of the manual, and on the simplicity of the methods' signature; (b) the Learnability depends on both the quality of the manuals and the complexity of the component's design, in particular on the ratio of words used to describe each interface or class, and on the percentage of methods with no return values; and (c) the Operability depends on the Complexity of the Design, in particular on a combination of the configurable parameters per method and the percentage of methods with no return values.

Our plans are now to carry out further experiments with more components (e.g., from other application domains) in order to gather more data that can help us further corroborate our results and findings, and refine our equations. Finally, we are packaging the tools we have developed for automating the measures (i.e., the

programs that analyze the component manuals, and those that "interrogate" the components using reflection) so we can provide soon an evaluation service to our local software industry for helping them select the components that best suit their applications with less effort and more precision.

References

[1] ISO/IEC 14598:2001. Software Engineering -- Product Evaluation. International Standards Organization, Geneva, Switzerland, June 2001.
[2] ISO/IEC 9126-1:2001. Software Engineering -- Product Quality -- Part 1: Quality model. International Standards Organization, Geneva, Switzerland, June 2001.
[3] ISO/IEC FDIS 25000:2005. Software Engineering -- Software Product Quality Requirements and Evaluation (SQuaRE) -- Guide to SQuaRE. International Standards Organization, Geneva, Switzerland, January 2005.
[4] Manuel F. Bertoa and Antonio Vallecillo. Quality attributes for COTS components. I+D Computacion, 1(2):128{144, November 2002.
[5] Pere Botella, Xavi Burgues, Jose Carvallo, Xavi Franch, and Carme Quer. Using quality models for assessing COTS selection. In Proc. of WER'02, pages 263{277, Valencia, Spain, November 2002.
[6] Allan W. Brown and KurtWallnau. The current state of CBSE. IEEE Software, 15(5):37--46, Sep-Oct 1999.
[7] S. Sedigh Ali, A. Ghafoor, and R.A. Paul. Software engineering metrics for COTS based systems. IEEE Computer, 34(5):44{50, May 2001.
[8] R. Simao and A. Belchior. Quality characteristics for software components: Hierarchy and quality guides. In Piattini, Cechich and Vallecillo, editors, Component-Based Software Quality: Methods and Techniques, number 2693 in Lecture Notes in Computer Science, pages 188{211, Heildelberg, 2003. Springer-Verlag.
[9] Hironori Washizaki, Hirokazu Yamamoto, and Yoshiaki Fukazawa. A metrics suite for measuring reusability of software components. In Proc. 9th Int'l Software Metrics Symposium (METRICS'03), pages 221{225, Sydney, Australia, September 2003. IEEE Computer Society Press.
[10] Manuel F. Bertoa, Jose M. Troya, and Antonio Vallecillo. A survey on the quality information provided by software component vendors. In Proc. of the 7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2003), pages 25{30, Darmstadt, Germany, 21 July 2003.
[11] IEEE Std 1061-1998, IEEE Standard for a Software Quality Metrics Methodology. IEEE Computer Society, 1998.
[12] Manuel F. Bertoa, Jose M. Troya, and Antonio Vallecillo. Measuring the Usability of Software Components. Submitted for publication, 2005.