Available online at www.sciencedirect.com
SCIENCE DIRECT®
The Journal of Systems and Software 79 (2006) 427–439
The Journal of Systems and Software
www.elsevier.com/locate/jss

Measuring the usability of software components

Manuel F. Bertoa *, José M. Troya, Antonio Vallecillo

Dpto. de Lenguajes y Ciencias de la Computación, Universidad de Málaga, ETSI Informatica. Campus Teatinos, 29071 Malaga, Spain

Received 9 March 2005; received in revised form 23 June 2005; accepted 23 June 2005
Available online 9 August 2005

Abstract

The last decade marked the first real attempt to turn software development into engineering through the concepts of Component-Based Software Development (CBSD) and Commercial Off-The-Shelf (COTS) components, with the goal of creating high-quality parts that could be joined together to form a functioning system. One of the most critical processes in CBSD is the selection of a set of software components from in-house or external repositories that fulfil some architectural and user-defined requirements. However, there is a lack of quality models and metrics that can help evaluate the quality characteristics of software components during this selection process. This paper presents a set of measures to assess the Usability of software components, and describes the method followed to obtain and validate them. Such a method can be re-used as a pattern for defining and validating measures for further quality characteristics.

Keywords: Software measurement; Software components; Usability measures

1. Introduction

Component-based software development (CBSD) has become an important alternative for building software applications, and in particular for distributed systems. This approach 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 pre-fabricated software 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.

One of the cornerstone processes of CBSD is the proper selection of the components. So far, most of the Software Engineering community efforts have concentrated on the functional aspects of components, leaving aside the (difficult) treatment of their quality and extra-functional properties. However, once the technical and technological issues of CBSD seem to have been successfully addressed, these kinds of properties deserve all the attention—software engineers have realized that they are essential for any industrial development process.

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 (2001) and the ISO/IEC 9126 (2001) series) are still under revision in year 2005. The SQuaRE project (ISO/IEC 25000, 2005) has been created specifically to make them converge, trying to eliminate the gaps, conflicts, and ambiguities that they 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 components and CBSD. In this sense, emergent

* Corresponding author. Tel.: +34 952132794; fax: +34 952131397.
E-mail addresses: bertoa@lcc.uma.es (M.F. Bertoa), troya@lcc.uma.es (J.M. Troya), av@lcc.uma.es (A. Vallecillo).

0164-1212/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2005.06.026

proposals in this area try to define quality models for software components and for component-based systems (Bertoa and Vallecillo, 2002; Botella et al., 2002; Brown and Wallnau, 1999; Ali et al., 2001; Simao and Belchior, 2003; Washizaki et al., 2003).

In recent works (Bertoa and Vallecillo, 2002; Bertoa et al., 2003), 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 (Bertoa and Vallecillo, 2002). Then, we looked at the information provided by software component vendors, analyzing the information they provide about the components they sell or license, with the purpose of determining how many of these metrics were in fact computable. We found that the information provided by vendors is normally scarce and mostly insufficient for any quality analysis (Bertoa et al., 2003).

The goal of our work is to define a set of objective, easy to automate, pre-defined measures for software components, that can be used to describe and assess their quality characteristics, and that can assist in their selection process. Such measures could provide information similar to that provided by the hardware components' catalogues, which are very helpful for selecting them and for looking for replacements if back-up or alternative solutions are required.

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 characteristics, 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.

The paper is organized as follows. After this introduction, Section 2 briefly presents CBSD and the basic software measurement concepts upon which proper definitions of measures can be given. Section 3 defines the concept of Usability, adapting this quality characteristic from the ISO/IEC 9126 Quality Model to the realm of software components. Then, Section 4 discusses the set of measurable elements of a software component that can be used to evaluate its Usability. Section 5 identifies the factors that influence this quality characteristic and defines a set of Usability measures. The process followed for identifying the appropriate component attributes, how the measures assess the different quality sub-characteristics of Usability, and the experiments performed to validate such measures is described in Section 6. Section 7 discusses some related works. The paper finishes by drawing some conclusions and outlining some future research activities in Section 8.

2. Basic concepts

2.1. CBSD

First of all, we need to define what we understand by a software component. In this paper we will adopt Szyperski's definition: software components are binary units of independent production, acquisition and deployment that interact to form a functioning system (Szyperski, 2002). The adjective "COTS" will refer to a special kind of components: commercial entities—i.e. that can be sold or licensed—that allow for packaging, distribution, storage, retrieval, and customization by users, which are usually coarse-grained, and that live in software repositories (Brown and Wallnau, 1999).

In the early 21st century there is a growing market for software components. As technology has matured, and component platforms (such as EJB, CORBA or .NET) provide effective solutions for building commercial component-based applications, there is a shift in industry towards trying to re-use third-party components, assembling them instead of building them from scratch—i.e. we are moving from a *development*-centric to a *procurement*-centric approach. Large organizations such as the US Army have already decided to move to commercial components in order to keep their software development costs and efforts under control (Sweeney et al., 2001). Software component vendors are proliferating as well, with companies such as ComponentSource which are starting to successfully distribute, sell, and license third-party components (Bertoa et al., 2003; Seppanen and Helander, 2001). Finally, major traditional software companies are also packaging their products as components (e.g., Sun, Microsoft, SAP, etc.).

However, as we mentioned earlier, the evaluation of the quality of software components is still an unresolved issue, and component selection is a subjective and cumbersome process. We need to be able to count on agreed quality models and sets of pre-defined measures in order to properly address these problems.

2.2. Software measurement concepts

In general, there is a lack of consensus about how to define and categorize software product quality characteristics. Here we will basically follow the terminology defined in ISO/IEC 15939 (2002).

At the highest level, the main goal of a software measurement process is to satisfy certain *information needs* by identifying the *entities* and the *attributes* of these *entities* (which are the targets of the measurement process). *Attributes* and *information needs* are related through *measurable concepts* (which belong to a *Quality Model*).

In this paper we will concentrate on one particular *information need*, namely "the evaluation of 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."

As *quality model* we have selected an adaptation of ISO/IEC 9126. Since that model is a generic quality model for any software product, it requires some customization for the particular case of software components. The customized model, denominated COTS-QM, was presented in Bertoa and Vallecillo (2002). As ISO/IEC 9126, COTS-QM defines a set of *characteristics* and *sub-characteristics*, together with the relationships between them. They provide the basis for specifying quality requirements and for evaluating the quality of components.

The *entities* of our study will be software components, no matter whether they are developed in-house, or acquired as COTS components. Examples of *attributes* of software components include their *size*, their *interface complexity*, or their *performance*, among others.

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.

Attributes are evaluated using *measures*. A *measure* relates a defined *measurement approach* and a *measurement scale*. A measurement approach is the logical sequence of operations, described generically, used in quantifying an attribute with respect to a specified scale (ISO/IEC 15939, 2002). A *measure* is expressed in *units*, and can be defined for more than one *attribute*. Examples of *measures* for software component *attributes* include the number of provided interfaces, the ratio of methods per required interface, or the throughput of video frames emitted per input video frame (they correspond, respectively, to possible *measures* for the aforementioned *attributes* size, interface complexity, and performance).

Three kinds of *measures* can be distinguished: *base measures*, *derived measures*, and *indicators*. Base measures do not depend upon any other measure (e.g., the number of figures in the manuals). A derived measure is derived from other base or derived measures (e.g., the ratio of methods per interface). An indicator is a measure that is derived from other measures using an *analysis model* (based on a *decision criteria*) to obtain a *measurement result* that satisfies an *information need* (e.g., the size of a component is "large" if it has more than 80 assemblies, provides more than 100 interfaces, and its manuals have more than 200,000 words).

The act of measuring software is a *measurement*, which can be defined as the set of operations that aims at determining a value of a *measurement result*, for a given *attribute* of an *entity*, using a *measurement approach*.

Please notice that other authors and proposals (including ISO/IEC 9126) use the term *metric* instead of *measure*. However, the definition of *metric* provided by both general and technical dictionaries do not reflect the meaning with which it is informally used in software measurement. In addition, the term *metric* is not present in the measurement terminology of any other engineering disciplines, at least with the meaning it is commonly used in software measurement. Therefore, the use of the term "*software metric*" seems to be imprecise, while the term "software measure" seems to be more appropriate to represent this concept. Accordingly, in the following we will use the term *measure*. This is also consistent with ISO/IEC and IEEE Computer Society positions which, in order to ensure both consensus and consistency with other fields of sciences, made a decision in the year 2002 to align their terminologies on measurement with the internationally accepted standards in this field. In particular, ISO-JTC1-SC7 is trying to follow as much as possible the ISO International Vocabulary of basic and general terms on Metrology (ISO VIM, 1993).

3. Usability in CBSD

As mentioned in the introduction, in this paper we will only focus on one of the six characteristics of ISO/IEC 9126, the Usability. The existing literature offers several definitions of Usability:

- The capability of the software product to be understood learned, used and attractive to the user, when used under specified conditions (ISO/IEC 9126-1:2000).
- The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use (ISO/IEC 9241-11:1998).
- The ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component (IEEE Std. 610.12-1990).
- Usability of a software product is the extent to which the product is convenient and practical to use (Boehm et al., 1978).

- The probability that the operator of a system will not experience a user interface problem during a given period of operation under a given operational profile (Fenton and Pfleeger, 1998).

Here, we will adopt the ISO/IEC 9126 definition, adapting it to the particular case of software components, according to COTS-QM (Bertoa and Vallecillo, 2002). Thus we define Usability as "the capability of a *software component* to be understood, learned, used and attractive to the *user*, when used under specified conditions".

The last sentence, "when used under specified conditions" (equivalent to "context of use" of ISO/IEC 9241-11) explicitly highlights that a product has no intrinsic usability, only a capability to be used in a particular context. In fact, the Usability is a quality characteristic that is intrinsically dependent upon the kind of "use" that is expected, and the kind of "user" that will use the product (the concept of user is implicit in the definition). Thus, we need to clarify the users and the kind of usage of the software components whose Usability we are trying to assess.

From our perspective, component *users* will be "the members of the software teams that develop and maintain a component-based system". Component users need to identify, locate and select them according to the system architectural constraints and the system owners' requirements, integrate them to build the system, and then proceed to the system maintenance when new component versions appear, or when components are upgraded for fixing problems. For brevity, in the following we will simply refer to such kinds of users as "system developers", although this term embraces in our case component-based system developers, component evaluators, system builders and integrators, and maintainers. Please notice that we have not included either the end-users of component-based systems, nor the developers of the individual components themselves. From our perspective, neither of them can be regarded as component users: end-users use the global system, but not the individual components; similarly, component developers fabricate the individual components, but they do not necessarily use them to build systems.

ISO/IEC 9126 and COTS-QM define Usability in terms of five sub-characteristics: Understandability, Learnability, Operability, Attractiveness, and Usability Compliance. In the case of software components, they can be defined as follows:

- *Understandability*: the capability of the component to enable the user to understand whether the component is suitable, and how it can be used for particular tasks and conditions of use. System developers should be able to select a component suitable for their intended use. For example, component elements (e.g. interfaces, operations) should be easy to understand.
- *Learnability*: the capability of the software component to enable the user (system developer) to learn its application. A Learnability measure should be able to assess the time and effort required by system developers to learn how to use particular functions (interfaces, operations, etc.), or the effectiveness of the documentation (manuals, help system, demos). For example, the user documentation and the help system should be complete, the help should be context sensitive and explain how to accomplish common tasks, etc.
- *Operability*: the capability of the software component to enable the user (system developer) to operate and control it. An Operability measure should be able to assess whether system developers can easily operate and control the component. Operability measures can be categorized by the dialogue principles described in ISO/IEC 9241-10:
  - suitability of the component for the task;
  - self-descriptiveness of the component;
  - controllability of the component;
  - conformity of the component with user expectations (requirements);
  - error tolerance of the component;
  - suitability of the component for individualization.
- *Attractiveness*: the capability of the software component to be attractive to the user.
- *Usability compliance*: the capability of the software component to adhere to standards, conventions, style guides or regulations relating to Usability. We do not know of any standards or regulations for component Usability yet, and thus we will not consider this sub-characteristic.

Attractiveness is specially difficult to assess in our particular case, since the kind of users that we consider here are not end-users, but system developers (integrators, maintainers, etc.). It is true that how attractive the component is does (subjectively) influence the rigor and attention expended on that component. It may well be the case that lesser components are given preferred selection treatment because of their attractiveness. However, this kind of (indirect) influence on the predisposition of the users for the component is difficult to measure, and quite different from the rest of the sub-characteristics in this case. Thus, we will not consider it in our study for the moment.

Summarizing, the Usability of a software component will be assessed in this work in terms of its *Understandability*, *Learnability* and *Operability*.

Once the quality sub-characteristics related to Usability are defined, we need to determine: (a) the set of *measurable concepts* that influence any of these sub-characteristics; (b) the component *attributes* that can

be measured in order to evaluate such measurable concepts, and how they affect each sub-characteristic; and (c) a set of *measures* (base measures, derived measures, and indicators) that evaluate such attributes. These measures need then to be validated in order to prove that they really provide useful information to assess the corresponding quality sub-characteristic(s).

Before we define any of those elements, and in particular the measures, we need to know the sort of information that is available about the entities we plan to assess (i.e., the software components). This information will be the only measurable elements based upon which measures can be defined.

4. Available information about software components

The fact that components are black-box and binary units of composition, whose internals cannot be viewed or accessed, only leaves us with their externally observable elements for evaluating their quality characteristics. In the case of Usability, these observable elements are just the documentation and some of the component's functional elements. Fig. 1 shows a UML class diagram with the information that we think is relevant to the Usability of a component, from all the information that is available for a commercial component.

In the following, and according to Fig. 1, the term *documentation* will refer to component manuals, demos, help system, and marketing information. By *functional elements* (FE) we will refer to the interfaces, operations, configurable parameters (e.g., public attributes) and events that a component may support or require from other components to achieve its functionality, i.e., to implement its services. Although some measures may be defined on individual elements (e.g., number of operations per interface), sometimes it is simpler to refer to the functional elements as a whole, independently of their granularity.

Of course, not all this information is available for all commercial components, as we discovered in our survey (Bertoa et al., 2003). However, we have tried to be realistic when defining the elements above, looking for a balance between the demands of information required by the measures, and the difficulty we know that software component vendors have for providing that information.

5. 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.

For instance, ISO/IEC 9126-1 defines a set of quality characteristics and sub-characteristics that constitute its Quality Model, and then ISO/IEC 9126 Parts 2–4 define, respectively, a set of (internal, external and in-use) measures (''metrics'' in ISO/IEC 9126 terminology) for measuring them. According to ISO/IEC 9126 itself, measures assess attributes, which influence one or

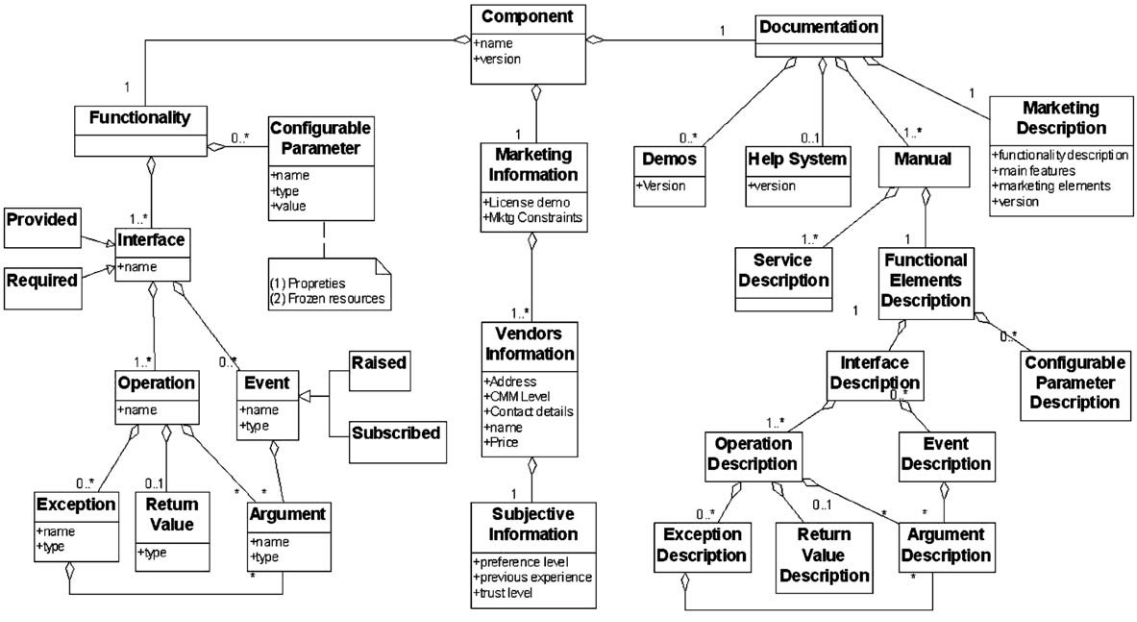

Fig. 1. Software component information relevant to Usability.

more quality sub-characteristics. However, the measures presented in Parts 2–4 of ISO/IEC 9126 are mostly derived measures, defined without any reference to the attributes they evaluate: they are just defined and assigned to quality sub-characteristics without any justification whatsoever. Moreover, ISO/IEC 9126 measures are assigned to just one quality sub-characteristic, although in theory one attribute may influence more than one quality characteristic (e.g., the "size" of a component usually affects both its maintainability and its understandability).

The situation is not better in the rest of the proposals, which do not consider attributes either. Worse than that, most of these proposals do not associate any quality characteristic to the measures they define—measures are just defined, but with no indication of either the attributes they evaluate or of the quality characteristics they try to assess.

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."

As previously mentioned, from the three measurable concepts related to software component Usability (*Quality of the Documentation*, *Complexity of the Solution* and *Complexity of the Problem*) we will just consider the first 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 Section 6.

Each attribute may have one or more measures (indicators, derived or base measures) that evaluate it. A summary of the measures initially defined for the *Quality of the Documentation* attributes is shown in Table 2, and the measures defined for those related to the *Complexity of Design* in Table 3. There should also be a last column in both tables with the base measures upon which derived measures are defined. This column has been omitted for space reasons and because most of these base measures are evident (for instance, the ratio of figures per kilo-word in manuals depends on two base measures: the number of figures and the number of words in the manual). Moreover, notice that many ratio measures repeat their denominator, which produces an unnecessary repetition of rows (this happens for instance with the number of methods or interfaces, which is used in many derived measures). In total, we initially defined 34 base measures and 35 derived measures.

Proposing a set of measures is just what most authors and standards do (Bertoa and Vallecillo, 2002; Botella et al., 2002; Brown and Wallnau, 1999; Ali et al., 2001; Simao and Belchior, 2003; Washizaki et al., 2003). But this is not enough. We now need to check that these measures:

(1) can be computed (i.e., we need to define their measurement approach);
(2) are objective and reproducible; and
(3) effectively asses the corresponding quality sub-characteristic (i.e., evaluate the degree with which they explain the Understandability, Learnability and/or Operability of a component).

In the first place, not all measures in Tables 2 and 3 were easy to compute. For example, the measure "Percentage of functional elements correctly used after read-

Table 1
Measurable concepts and attributes related to Usability

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 Mktg Info	Contents of Mktg Info
		Design complexity		Design legibility
				Interface Understandability
				Learning facility
				API complexity
				Customisability

Table 2
Measures related to quality of documentation

Attribute	Indicator	Derived measure
Contents of manuals	Manuals coverage	Percentage of functional elements (FE) described in manuals Percentage of interfaces described in manuals Percentage of methods described in manuals Percentage of configurable parameters described in manuals
	Manuals consistency	*Percentage of FE incorrectly described in manuals* Component version difference
	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 FE correctly used after reading the manuals*
Contents of demos	Demos coverage Demos consistency	*Percentage of FE included in demos* *Demo version difference*
Contents of Mktg Info	Mktg Info coverage Mktg Info consistency	*Number of FE in Mktg Info* *Mktg Info version difference*

Table 3
Measures related to complexity of design

Attribute	Indicator	Derived measure
Design legibility	Meaningful names	Percentage of functional elements (FE) with long names (>20 chars) Average length of FE names
Interface Understandability	FE Understandability	*Percentage of FE used without errors*
Learning facility	Time to use Time to expertise	*Average time to use correctly the component* *Average time to master the component*
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 configurable parameters per interface Ratio of configurable parameters per method

ing the manuals'' is very difficult to calculate. These measures are shown in *italics* in these tables. On the contrary, other measures which are difficult to compute for general software products can be easily obtained, and even automated, in the case of software components: using reflection (Maes, 1987) techniques we can interrogate the component and obtain information about its functional elements (interfaces, operations, fields, etc.). Similarly, we can make use of the fact that the documentation is mostly in electronic format to thoroughly analyze it. Thus, from our original set of measures, the ones that can be computed in an objective and reproducible manner are those shown in the left column of Tables 4 and 5. They add up to 19 base measures and 21 derived measures.

6. Validating the measures

6.1. Experimental validation

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)

Table 4
Correlation values (R) for derived measures

Derived measure	D_Usab	I_Usab	O_Usab	O_Und	O_Learn	O_Oper
Percentage of FE described in manuals	0.04	0.29	0.61	0.47	0.63	0.57
Percentage of interfaces described in manuals	0.30	0.18	0.32	0.28	0.00	0.05
Percentage of methods described in manuals	−0.21	−0.06	0.35	0.41	0.38	0.28
Percentage of configurable parameters 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 HTML files of manuals 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	−0.29	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 configurable parameter	−0.05	0.46	0.56	0.10	0.63	0.63
Percentage of FE with long names (>20 chars)	0.17	0.27	−0.10	−0.10	−0.05	−0.10
Ratio of interfaces per required 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 configurable parameters per interface	−0.15	−0.32	−0.03	0.22	0.09	0.13
Ratio of configurable parameters 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 5
Correlation values (R) for base measures

Base measure	D_Usab	I_Usab	O_Usab	O_Und	O_Learn	O_Oper
Number of words in manuals	−0.05	−0.08	0.08	0.12	0.38	0.44
HTML files of manuals	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 configurable 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 FE	−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 configurable parameters 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 (>20 chars)	0.22	0.01	−0.30	−0.04	0.02	0.04
Number of methods without return value	0.05	−0.13	−0.10	0.07	0.25	0.31
Number of methods without arguments	−0.07	−0.14	−0.10	−0.03	0.25	0.32

about the Understandability, Learnability and Operability of a set of software components:

(a) as *perceived* by a set of users according to the definitions of Understandability, Learnability and Operability, that were given to them;
(b) as *perceived* by a set of users when asked "indirectly" about the Understandability, Learnability and Operability of a set of components;
(c) *objectively*, as a result of a questionnaire that asked them to perform a set of tasks with the components, and evaluated the responses according to the time required to answer the questions and the correctness of the answers.

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 or too complex, and because there were enough candidate components to constitute a representative sample. All candidate components were products available from commercial vendors, advertised

in component repositories such as ComponentSource. They used different component models and technologies: Java, ActiveX and .NET.

The first experiments were more focused on the subjective evaluation of the Usability of the selected components, in order to get some figures about their Understandability, Learnability and Operability. Basically, users were asked to assign a value from a semantic scale of seven values (that was later translated into an integer value between −3 and 3) to a number of questions, depending on their subjective judgement. The last two experiments introduced more objective questions on whether some specific tasks and services were provided by the component or not, and how to perform them. Evaluation in this case was made depending on the time required to answer each question, and its correctness. A number between −3 and 3 was also assigned to each response.

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 Málaga, although one experiment was replicated in the University of Castilla-La Mancha.

- The *perceived Usability* was *directly* evaluated by providing users with the definition of Usability and of its three sub-characteristics, and they were asked to evaluate the degree with which the component fulfilled these definitions.
- The *perceived Usability* was also *indirectly* evaluated by asking users some questions that provided information about the Understandability, Learnability and Operability of the components, but without asking them about those sub-characteristics directly. For example, whether they thought that the services provided by the component were easy to understand or not; or if they considered that the component was easy to operate or not. These questions were asked based on the components' documentation, demos, etc., which is the only information that a component evaluator can count on for performing the selection process.
- Finally, we identified a set of common tasks and defined a questionnaire that requested users to check whether the component could accomplish some of them or not, and also how to perform them. Please notice that the results of these questions represented the *objective* evaluation of the Usability of the components, in contrast to the previous questionnaires that captured just the Usability, as *perceived* by the users. Hence, no matter what the subjective evaluation was, the last tests reflect the *real* component Usability, and therefore their figures were used as the basis for our statistical analysis.

6.2. Direct results from the experiments

From all these experiments we obtained six variables (whose values ranged between −3 and 3) 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 last experiment's questionnaire. Finally, the Objective Usability is the average of the other three objective values.

We found that there is a correlation between the perceived and objective Usability. In fact, the indirect perceived usability (**I_Usab**) and the objective usability (**O_Usab**) are correlated with $R = 0.85$.

Figs. 2 and 3 show the results obtained for the 12 sampled components. Components are listed ordered according to their **O_Usab** values. Component names have been omitted (they are just called C005, C012, etc.) in order to respect their privacy.

Once we had quantified the information about the Usability of the sampled components, the next step was to measure the components using our defined measures, and then to look for correlations between the measurement results obtained and the Usability figures. A strong correlation (e.g., $R > 0.95$) means that a measure explains well the Usability figures, and then demonstrates that the measure is representative and therefore valid for measuring the corresponding sub-characteristic. (A correlation coefficient R greater than 0.95 means that a measure explains $R^2 = 0.95^2 \approx 90\%$ of the Usability figures.) Tables 4 and 5 show, respectively, the correlations matrixes obtained for our base and derived measures.

Many interesting conclusions can be drawn from these tables:

- The ratio of manual (HTML) files per FE have a good correlation ($R = 0.93$) explaining an 86% of the component usability. Basically, this means that the structure, organization, and navigability of the electronic manual pages (each page is stored in an HTML file) influences the component Usability. In particular, HTML pages should not be very long and should not contain too much information in order to improve that Usability.
- The number of words per method (analogously, the number of words per FE) seems to explain well the

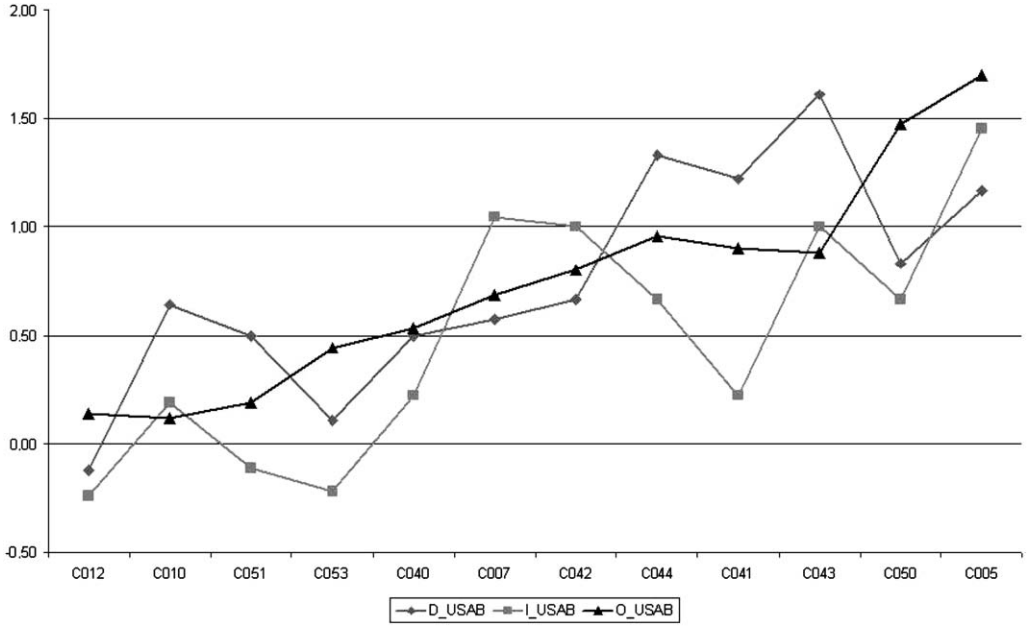

Fig. 2. Perceived and objective usability figures for the sampled components.

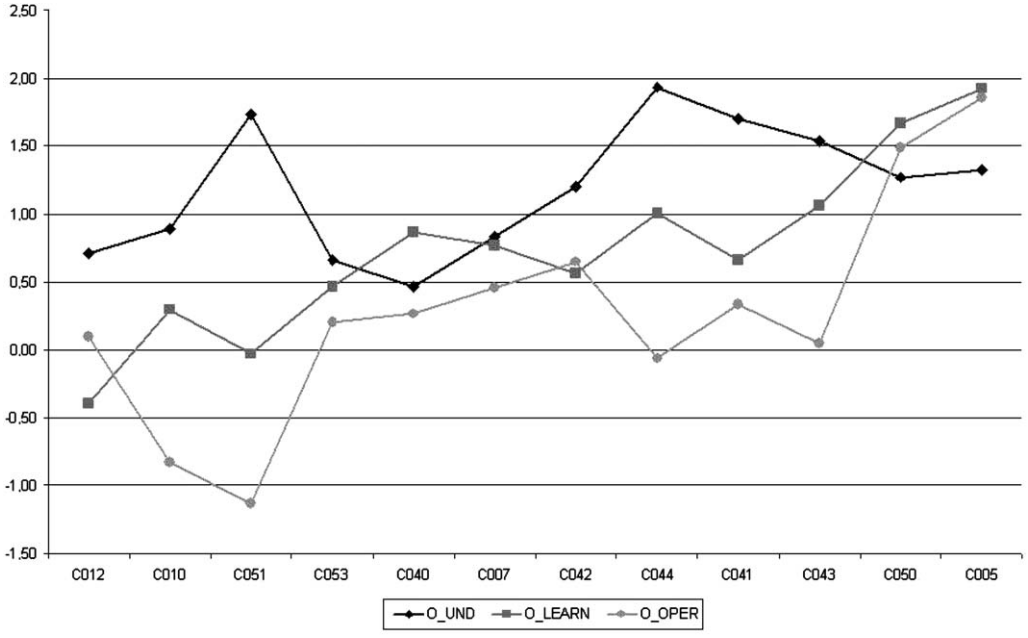

Fig. 3. Understandability, Learnability and Operability objective figures obtained for the sampled components.

component Learnability and Operability ($R^2 = 0.83$). This is consistent with the intuitive idea that a component is easier to learn and operate if its manuals contain abundant information about its methods and FE.

- Only one derived measure (the ratio of return values per method) has some influence on the component Understandability, although without any major statistical significance ($R = -0.82$, $R^2 = 0.67$).
- None of base measures are representative, being the highest value $R = 0.68$ for the number of HTML files of the manuals.

We also found some unexpected results:

- For instance, the number of figures and tables in the manuals does not seem to have a direct influence on the perceived Usability of the components. This sug-

gests that users do not appreciate the drawings and tables in the manuals when subjectively evaluating the component Usability (although this measure becomes influential for evaluating the objective Understandability, as we shall later see).
- We originally thought that the completeness of the manuals was going to be very important for the Usability. However, the percentage of classes, methods, and FE that are described in the manuals do not seem to have a strong influence on any of the Usability values. (Even when some of the manuals only describe 50% of the component's public classes and methods!)
- Finally, we also thought that the length of the names given to classes, methods, attributes, etc., might influence the Learnability and Understandability of a component. However, these impressions have not been corroborated by the figures obtained.

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., $R^2 \approx 1$) of the Understandability, Learnability, or Operability of a component, or of its Usability as a whole.

6.3. Linear regression analysis

At this point, we need to recall what we identified in the introduction as a common shortcoming of most software measurement proposals: the measures defined by most authors and International Standards are usually assigned to just one quality sub-characteristic, 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 greater than 0.97. These combinations are shown in Table 6 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.

In summary, all these sub-characteristics can be explained as a combination of one measure related to the *Quality of the Manuals* and another related to the *Complexity of the Design*. Hence, we can see how these two measurable concepts have influence on the three Usability sub-characteristics (on more or less degree, of course).

Please notice as well that some of the equations in Table 6 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 measures themselves, and than any other individual measure.

Notice as well that these results are fully consistent with the definitions of the three quality sub-characteristics, and also make perfect sense with our intuition. For instance, the controllability of the component and its suitability for individualization were two of the Operability requirements (Section 3). This is normally achieved in software components by means of configurable parameters. What we have precisely discoveredis that the size of the documentation on the configurable parameters has a strong influence on this

Table 6
Usability sub-characteristics and the measures that explain them

Sub-characteristic	Influential combination of measures		R^2	Relationship
Understandability (**O_Und**)	Ratio HTML files per FE (Fil)	Ratio of return values per method (RVpM)	0.970	**O_Und** = 0.200Fil − 1.423RVpM + 1.598
Learnability (**O_Learn**)	Kilo-words per method (WpM)	Ratio of arguments per method (ApM)	0.973	**O_Learn** = 1.789WpM − 2.090ApM + 1.712
Operability (**O_Oper**)	Kilo-words per configurable parameter (WpCP)	Ratio of return values per method (RVpM)	0.980	**O_Oper** = 0.804WpCP − 8.265RVpM + 3.486

sub-characteristic (when combined with the percentage of methods with no return values).

7. Related work

Our work can be related to two major lines of research: the definition of measures for software components, and the evaluation of the Usability of software products and artifacts.

We have already mentioned the international standards (ISO/IEC 9126, 2001; ISO/IEC 15939, 2002) and research proposals (Bertoa and Vallecillo, 2002; Botella et al., 2002; Brown and Wallnau, 1999; Ali et al., 2001; Simao and Belchior, 2003; Washizaki et al., 2003) that provide sets of measures for software components. In fact, some of the measures discussed in this paper have been borrowed (with slight adaptations in some cases) from these sources. However, we also found that many of the proposed measures were very difficult or impossible to compute, and none of them seem to have been validated. Furthermore, as mentioned in the introduction, they are assigned to just one quality sub-characteristic. Our validation experiments have clearly showed how measures normally influence more than one sub-characteristic, as intuition has always suggested. In addition, none of the proposals analyze the degree of influence of each measure on the corresponding quality sub-characteristic. In this sense, our work has served to evaluate such a degree of influence, and also to select a small group of very representative measures, discarding all those with not enough influence.

Other works (e.g., Bass and John, 2003; Folmer and Bosch, 2004) discuss the quality characteristic of Usability—but in the context of Software Architecture (SA). These works propose a set of techniques and recommendations for improving the Usability of SA, but without providing any kind of measure for this quality characteristic. In this sense, our work can be seen as complementary to those because it can help defining similar measures for SA that objectively assess how their suggested techniques work in practice.

Finally, there are other important works on the Usability of user interfaces and human–computer interactions, specially those from Shneiderman (2000), Shneiderman (2004) on *Universal Usability*, and from Nielsen (2004). Despite their relevance to any serious study on Usability, these works are very general and more focused on the access to information and how to display it for Usability, so issues such as technology variability, user diversity and gaps in user knowledge can be successfully tackled. Our study is however focused on a particular domain (component selection for CBSD), with a concrete kind of software products and users, and where these users have homogeneous profiles. Moreover, the way in which the information is accessed and displayed in our domain tends to be quite similar for all software components, apart from the differences captured by some of our measures (organization and navigability of the manuals, basically).

8. Conclusions

This paper has presented a set of base and derived measures that can be used to evaluate the Usability of software components. In addition, we have shown how to validate these measures, calculating both how they individually influence the ISO sub-characteristics for Usability, and also the most representative combinations of measures that can effectively assess these sub-characteristics.

We have seen how the appropriate combinations of 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 method's signature (in particular, the percentage of methods with no return values); (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 method or operation, and on the number of arguments per method; and (c) the Operability mainly depends on a combination of the amount of information available about the component configurable parameters, 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. Based on these equations, we also plan to work on analysis models that can help us define the appropriate quality indicators for Usability. 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.

Acknowledgements

The authors would like to thank Dr. Carmen Morcillo for her guidance and help with our statistical

analysis. We would also like to acknowledge our colleagues Ricardo Conejo, Coral Calero, Antonio Maña and Francisco Durán for their support with the experiments. Finally, we want to thank the reviewers for their insightful and constructive comments, that greatly helped improve earlier versions of this paper.

The work presented here has been funded by Spanish Research Project TIC2002-04309-C02-01.

References

Ali, S., Ghafoor, A., Paul, R., 2001. Software engineering metrics for COTS based systems. IEEE Computer 34 (5), 44–50.

Bass, L., John, B.E., 2003. Linking usability to software architecture patterns through general scenarios. The Journal of Systems and Software 66 (3), 187–197.

Bertoa, M.F., Troya, J.M., Vallecillo, A., July 2003. A survey on the quality information provided by software component vendors. In: Proceedings of the 7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2003). Darmstadt, Germany, pp. 25–30.

Bertoa, M.F., Vallecillo, A., 2002. Quality attributes for COTS components. I+D Computación 1 (2), 128–144.

Boehm, B., Brown, J.R., Kaspar, J., et al., 1978. Characteristics of Software Quality. North Holland, Amsterdam.

Botella, P., Burgués, X., Carvallo, J., Franch, X., Quer, C., Nov 2002. Using quality models for assessing COTS selection. In: Proceedings of WER'02. Valencia, Spain, pp. 263–277.

Brown, A.W., Wallnau, K., 1999. The current state of CBSE. IEEE Software 15 (5), 37–46.

Fenton, N., Pfleeger, S., 1998. Software Metrics: A Rigorous Approach, second ed. Chapman & Hall, London.

Folmer, E., Bosch, J., 2004. Architecting for usability: A survey. The Journal of Systems and Software 70 (1), 61–78.

ISO VIM, 1993, second ed.International Vocabulary of Basic and General Terms in Metrology International Standards Organization, Geneva, Switzerland.

ISO/IEC 14598, 2001. Software Engineering—Product Evaluation. International Standards Organization, Geneva, Switzerland.

ISO/IEC 9126, 2001. Software Engineering—Product Quality—Part 1: Quality Model. International Standards Organization, Geneva, Switzerland.

ISO/IEC 15939, 2002. Software Engineering—Software Measurement Process. International Standards Organization, Geneva, Switzerland.

ISO/IEC 25000, 2005. Software Engineering—Software Product Quality Requirements and Evaluation (SQuaRE)—Guide to SQuaRE. International Standards Organization, Geneva, Switzerland.

Maes, P., 1987. Concepts and experiments in computational reflection. Proceedings of OOPSLA'87: Conference on Object-Oriented Programming Systems, Languages and Applications. ACM Press, Orlando, FL, pp. 147–155.

Nielsen, J., 2004. Designing Web Usability: The Practice of Simplicity. New Riders Publishing, Indianapolis.

Seppanen, V., Helander, N., 2001. Original software component manufacturing: Survey of the state of the practice. Proceedings of the 27th Euromicro Conference. IEEE Computer Society Press, Warsaw, Poland, pp. 138–145.

Shneiderman, B., 2000. Universal usability: Pushing human–computer interaction research to empower every citizen. Communication ACM 43 (5), 84–91.

Shneiderman, B., 2004. Designing the User Interface: Strategies for Effective Human–Computer Interaction, fourth ed. Addison-Wesley, Reading, MA.

Simao, R., Belchior, A., 2003. Quality characteristics for software components: Hierarchy and quality guides. In: Piattini, M., Cechich, A., Vallecillo, A. (Eds.), Component-Based Software Quality: Methods and Techniques. No. 2693, Lecture Notes in Computer Science. Springer-Verlag, Heildelberg, pp. 188–211.

Sweeney, L., Kortright, E., Buckley, R., July 2001. Developing an RM-ODP based architecture for the defense integrated military human resources system. In: Cordeiro, J., Kilov, H. (Eds.), Proceedings of the Workshop On Open Distributed Processing: Enterprise, Computation, Knowledge, Engineering and Realisation (WOODPECKER'01). Setubal, Portugal, pp. 110–124.

Szyperski, C., 2002. Component Software, second ed.Beyond Object-Oriented Programming Addison-Wesley, Cambridge, MA.

Washizaki, H., Yamamoto, H., Fukazawa, Y., 2003. A metrics suite for measuring reusability of software components. Proceedings of 9th Int'l Software Metrics Symposium (METRICS'03). IEEE Computer Society Press, Sydney, Australia, pp. 221–225.