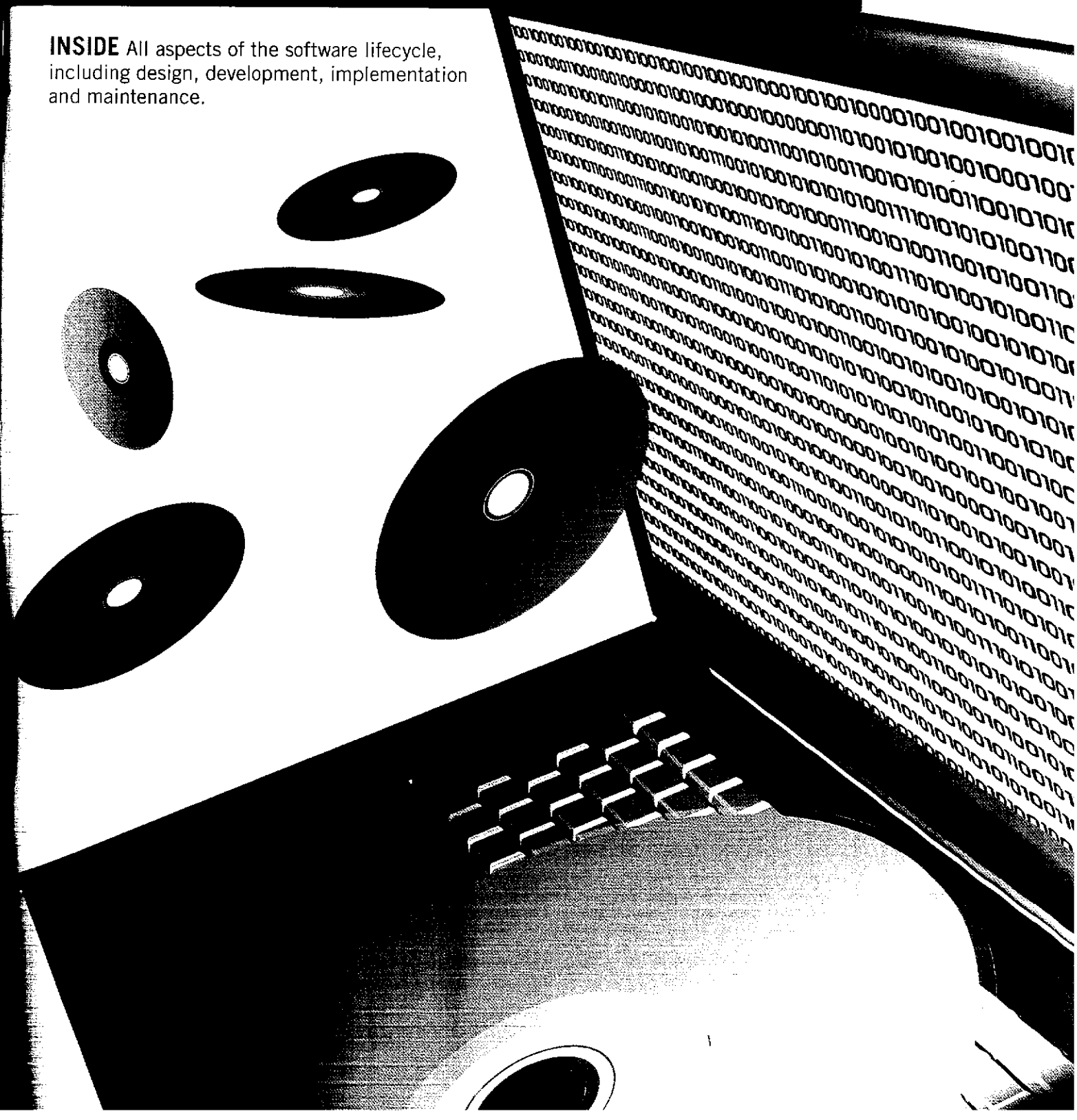
IET Journals

Vol 4 | Issue 1 | ISSN 1751-8806
March 2010

IET Software

INSIDE All aspects of the software lifecycle, including design, development, implementation and maintenance.



Published in IET Software
Received on 22nd July 2008
Revised on 16th June 2009
doi: 10.1049/iet-sen.2008.0056

ISSN 1751-8806

Improving interpretation of component-based systems quality through visualisation techniques

M.A. Moraga[1] *C. Calero*[1] *M.F. Bertoa*[2]

[1]*Alarcos Research Group, Institute of Information Technologies & Systems, University of Castilla-La Mancha, Paseo de la Universidad, Ciudad Real 4-13071, Spain*
[2]*Department Lenguajes y Ciencias de la Computación, University of Málaga, Complejo Tecnológico, Campus Teatinos, Málaga 29071, Spain*
E-mail: MariaAngeles.Moraga@uclm.es

Abstract: Component-based software development is increasingly more commonplace and is widely used in the development of commercial software systems. This has led to the existence of several research works focusing on software component-based systems quality. The majority of this research proposes quality models focused on component-based systems in which different measures are proposed. In general, the result of assessing the measures is a number, which is necessary to determine the component-based system quality level. However, understanding and interpreting the data set is not an easy task. In order to facilitate the interpretation of results, this study selects and adapts a specific visual metaphor with which to show component-based systems quality. A tool has additionally been developed which permits the automatic assessment of the measures to be carried out. The tool also shows the results visually and proposes corrective actions through which to improve the level of quality. A case study is used to assess and to show the quality of a real-world component-based software system in a graphic manner.

1 Introduction

With the growing emphasis on reuse, the software development process is moving towards component-based style [1], namely component-based software development (CBSD). CBSD is a software development methodology proposed by Szyperski [2] which tackles the complexity of growing software systems. This new methodology is focused on assembling previously existing components in larger software systems, and migrating existing applications towards component-based software systems [3]. The software components, which are integrated into the system, can be defined as binary units of independent production, acquisition and deployment which interact to form a functional system [2]. In the context of this methodology the evaluation and acquisition process of software components, whose objective is to choose the best component to be integrated into the system, is essential. Besides functional characteristics, the acquisition process should also take into account the non-functional characteristics of the component, that is, the characteristics related to its quality. It may be assumed that if the software components, which are integrated into the system, have a good level of quality then the system will also have a good level of quality. However, the quality of a software system not only depends on the quality of its individual components but also on the interdependence among them [4]. It is therefore necessary to study the quality of component-based software systems. Literature shows various quality models which are specific to this kind of systems. However, the majority propose a set of measures whose result is a number. For humans, the interpretation of a set of numbers (i.e. the numbers resulting from assessing measures) is not an easy task. Visualisation techniques can therefore be used to facilitate the comprehension of these measures. Visualisation combines data pre-processing and

data representation with the human capability of analysing graphics. The resulting visual display enables the scientist or engineer to visually perceive features which are hidden in the data but which are nevertheless needed for data exploration and analysis [5]. From our point of view, visualisation techniques can be used to represent certain measures or parameters of CSBD, in an attempt to detect their quality level with ease. The use of visualisation in component-based software systems permits a simple and rapid understanding of the system's quality features only if the chosen visual metaphor is suitable and well defined. An adequate visual metaphor with which to correctly visualise the quality of a software component-based system is therefore needed.

The objectives of this paper are the following. Having taken into account some of the existing measures for component-based software systems and the features that we consider that a measure should meet, a set of measures which are useful for component-based software systems are first identified. Secondly, a visual metaphor is selected from literature and is adapted to represent the measures identified in order to assess the quality of a component-based software system. Finally, the tool that we have developed to automatically assess and visualise the measures is presented. Furthermore, a case study has been carried out using a real-world component-based software system.

This paper is structured as follows. In Section 2, the main proposals relating to visualisation and component-based software systems quality are presented. Section 3 presents the set of identified measures with which to assess component-based software systems quality, while Section 4 shows the visual metaphor which has been selected and adapted to represent them. In Section 5, the tool developed to support this is presented. In Section 6, a case study is carried out. Finally, our conclusions and future work are outlined in Section 7.

2 Related works

The main goal of this section is to give an overview of the existing proposals related to the issues with which this paper is concerned: component-based software systems quality and visualisation.

2.1 Component-based software systems quality

Among the advantages of software development using component-based technology the following can be highlighted [6]:

- It reduces time to market by shifting developer resources from component-level development to integration
- It reduces costs.
- It facilitates rapid incremental delivery, thus allowing developers to release modules as they are integrated and offer product upgrades as various components evolve.

However, certain disadvantages can be also identified, such as integration difficulties, performance constraints and incompatibility among products from different vendors [6]. Consequently, the integration of various software components, which satisfy a certain quality level, does not signify that the resulting system achieves the same quality levels. Several quality models which are specific to component-based software systems have therefore been defined. Nevertheless, it should be noted that, if our objective is taken into consideration (visualising the quality of a component-based system), then a quality model is not sufficient. Proposals which identify measures in order to assess the component-based system quality are needed. Some examples of this can be found in literature. One example is [4], which proposes measures to estimate quality of service based on data collected while testing the component. The component execution graph (CEG) is mapped into a homogeneous continuous Markovian process, in order to define analytical methods with which to evaluate the make-span, cost and reliability of CEG. The CEG graph can effectively model real-world component-based software systems. A further example is [6], in which the proposed measures are divided into three categories: management, requirements and quality. Among the measures identified for quality are: adaptability, complexity of interfaces and integration, integration test coverage, end-to-end test coverage, fault profiles, reliability and customer satisfaction. When we attempted to make use of the defined measures to asses the quality of component-based software systems, we encountered several important problems. For example, many measures are ill-defined and incomplete, and the vocabulary and terminology used in the various measurement proposals are inconsistent and conflicting. These problems derive from the fact that the software measurement field is a relatively young discipline which is still evolving and has not yet fully matured. Therefore the terminology used, such as measure, measurement or attribute does not have a uniform definition which is accepted by all researchers and practitioners, and the term 'metrics' is used only in the context of software measurement whereas others scientific areas use the term 'measure'.

According to García *et al.* [7] inconsistencies and conflicts in terminology also appear among the standards of different organisations such as ISO, IEEE and IEC. Moreover, they affirm that there are also inconsistencies among different standards belonging to the same organisation. With regard to ill-defined measures, it should be highlighted that some authors do not provide all the necessary information of the measure. For example, Table 1 shows a non-exhaustive study [8] of certain measures proposed by various authors. This allows us to identify the typical problems mentioned: ambiguity in definition, inadequate formalism and insufficient validation.

All of the aforementioned factors impede the automation of the measures since it is impossible to understand what exactly each measure means. In other cases, the measures are hard to calculate or their automation may not even be possible (what they are supposed to measure cannot be calculated in an automated process). In addition, some measures are subjective, and it is necessary to know the user's opinion if they are to be assessed. Moreover, some authors do not say how to assess the measure, and provide only a name and a definition. All this makes the automation of their measures difficult.

2.2 Software visualisation proposals

According to Gershon [5], visualisation is more than simply a method of computing. Visualisation is the process of transforming information into a visual form, thus enabling users to observe that information. The resulting visual display enables the scientist or engineer to visually perceive features which are hidden in the data but which are nevertheless needed for data exploration and analysis.

The goal of software visualisation is not to produce neat computer images, but rather computer images which evoke mental images through which to better comprehend software [9]. In order to achieve this, metaphors can be used.

Several definitions of the term metaphor exist. In our research we consider, in accordance with [10], that a visual metaphor can be defined as the mechanism used to represent information graphically, due to the non-inherently physical nature of that information.

Therefore the most important issue in the field of visualisation is that of the visual metaphor which is used to represent information, as this is the key to a correct understanding of the information that is being visualised.

The eight most frequently used means of visualisation are: bar charts, matrix views, landscapes, network views, scatterplots, histograms, data sheets and time tables [11]. Table 2 provides a summary of the main characteristics of these visual metaphors.

With regard to software visualisation tools, one of the most remarkable is that presented in [12] since it has been specifically designed for software visualisation, focusing on object-oriented systems (in this metaphor each class is represented by a box). We therefore believe that the general ideas presented in this proposal may be useful for the component-based development domain. In [13], the authors propose a three-dimensional (3D) representation for the visualisation of large software systems (i.e. to represent source code and related attributes). In [14] an approach to assist in the understanding of system evolution, based on the definition of a graphical matrix displaying classes that are enriched with metrics information is presented.

For static program visualisation we should mention the GRASP tool whose data input is the program code and which produces control-structure diagrams [15], the aiSee tool [16] which generates control-flow graphs and the GOOSE [17] and VizzAnalyzer [18] tools which are concentrated on class and methods.

The JaVis [19] tool traces the execution of concurrent programs and detects deadlock situations. The TANGO tool [20] is able to represent animations of state transitions.

In the context of software evolution, the GEVOL tool is a system that visualises the evolution of software using a novel graph drawing technique for the visualisation of large graphs with a temporal component [21].

However, despite the existence of several tools with which to show software visualisation metaphors for different contexts (see above), there are no proposals for CBSD.

3 Measures for component-based software systems

In order to choose a set of measures which assess component-based software system quality, two requirements are necessary. On the one hand, the measure must be well defined, and according to McGarry *et al.* [22], should have a name, a description, a type of measure, a measurement method, a scale and a unit of measurement. On the other hand, the measure should be calculated automatically.

However, most of the measures for the quality of CBSD systems that can be found in literature do not meet the aforementioned requirements. We have, therefore, decided to define a set of measures based on those which already exist.

Table 1 Proposals where measures are not defined or validated

Author	Definition of the measure	Is there validation?
Simão and Belchior [23]	no	no
Bertoa and Vallecillo [24]	informal	no
Sedigh-Ali *et al.* [25]	no	no
Gill and Grover [26]	no	no
Dumke and Schmietendorf [27]	informal	descr. stats.
Boxall and Araban [28]	mathematical	descr. stats.
Washizaki *et al.* [29]	mathematical	regresion model

Table 2 Comparative table of the different visual metaphors

Metaphor	Attributes	Minimum element size	Advantages	Limiting factors
bar chart	2 (width, height)	1 pixel/bar	simplicity	small number of attributes. High quantity of information, low legibility
		1 pixel/separation		
matrix view	1–4 (colour, texture, size and shape)	10 × 10 pixels/cell	vast quantity of information (13 000)	aspect ratio
landscapes	3 (height, width and colour)	several hundred pixels/ three-dimensional elements	vast quantity of information (13 000)	occlusion of short bars at the back. Aspect ratio
network view	3 (size, node colour and link width)	depends on information.	individual characteristics and relationships	links connecting overplot
		Minimum width 1 pixel/ link		
ScatterPlot	3 (colour, *X*- and *Y*-axis)	3 pixels diameter	information evolution and trend predictions	points overplot
histograms	3 (smoothed colour, *X*- and *Y*-axis)	1 pixel/point	information evolution and trend predictions	complexity of calculation.
				Points overplot
data sheets	tens	depends on text characteristics	vast quantity of sorted information (rows)	not a graphic representation
ParaBox	4 (shape, shape colour, line type and line colour)	depends on information	vast quantity of lines and tens of rows	lines overplot
time tables	unknown	1 pixel/mark	vast quantity of marks organised into hundreds of categories	points overplot. Only useful for time-stamped information

We should stress that we have worked with a company in order to identify the relevant quality attributes. This company develops both software components and software component-based systems. The company's workers therefore have to deal with different problems related to software component quality on a day-to-day level. We have used their knowledge in order to detect both the important quality attributes and the measures which must be visualised when using the tool. Various meetings between the developers of software component-based systems and researchers were thus scheduled. The initial objective was to determine the characteristics of software component-based systems which were important for the developers (from the point of view of their quality). Next, the researchers chose and proposed a set of measures in order to assess the relevant characteristics. Finally, the proposed measures were analysed by the developers who accepted them with some changes. Our proposal thus has the advantage of using measures that are genuinely relevant in the industrial field.

A further interesting aspect of this proposal is that the software measurement ontology (SMO) has been used in order to define our measures in a non-ambiguous manner. A detailed description of SMO is available in [7].

Bearing all this in mind, the set of proposed measures can be classified into three groups according to their level of application.

1. Measures for component quality: This group of measures attempts to capture the components' quality when they are integrated into a particular system. The following measures are classified in this group: functionality of the component, component coverage, component use and weighted component quality.

2. Measures for system utilisation: These measures are focused on the system, and not on particular components. Two measures have been defined: system coverage and level of use.

3. Measures for system integration: Finally, a measure has been used to indicate the degree of difficulty involved in integrating the components into the system (the Lines of Glue Code).

The classification of these measures is shown in Fig. 1.

We shall now present the different measures for each group. It should be noted that two different types of measures have been identified: base measures (measures that do not depend

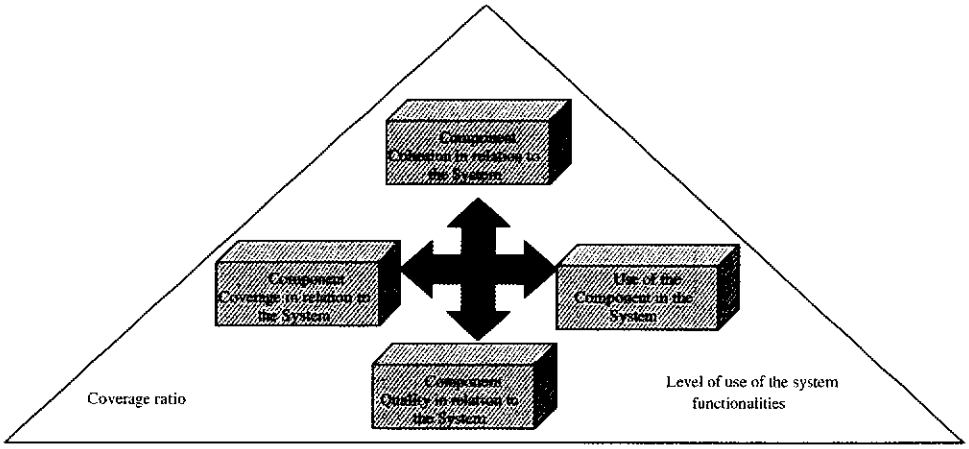


Figure 1 *Classification of the measures*

upon any other measures) and derived measures (measures which are derived from other base or derived measures).

As some measures are used in more than one derived measure, we shall first present them.

3.1 Common measures

1. *Measure name*: Services provided by component

• *Description*: This measure indicates the total number of services provided by a component. The system can use the component through these services.

Type of measure	Measurement method	Scale	Unit of measurement
base	number of services provided by a component	integer > 0	services

2. *Measure name*: Average number of services (AoS)

• *Description*: This indicates the AoS which are provided by all the components of the system.

Type of measure	Measurement method	Scale	Unit of measurement
derived	$\sum \text{SpC}_i / \sum i$ where i is a component	real number > 1	services/ components

3. *Measure name*: Used services of a component

• *Description*: This indicates the total number of services of a component which are used by the system.

Type of measure	Measurement method	Scale	Unit of measurement
base	number of used services of a component	integer ≥ 0	services

4. *Measure name*: Minimum calls of a component

• *Description*: This indicates the minimum number of calls to the services provided by a component.

Type of measure	Measurement method	Scale	Unit of measurement
base	number of calls to a service of the component	integer ≥ 0	service calls

5. *Measure name*: Used services of the system

• *Description*: This measure counts the total number of services used by the system or by the components which are integrated into the system.

Type of measure	Measurement method	Scale	Unit of measurement
base	number of services used in the system	integer ≥ 0	services

6. *Measure name*: Minimum calls in the system

• *Description*: This measure indicates the minimum number of calls to the components which are integrated into the system.

Type of measure	Measurement method	Scale	Unit of measurement
base	number of minimum calls to a service of a component	integer > 0	service calls

3.2 Component quality measures

The information related to the measures which are focused on the component quality in relation to the system (these are situated in the boxes in Fig. 1) as follows

3.2.1 Functionality of the component

7. *Measure name*: Functionality of the component (FoC)

• *Description*: This measure compares the number of services that a component offers with the AoS which are offered by all the components in the system. Components with an average number of offered services are around the value of 1. Components with few or many offered services are close to the value of 0.

Type of measure	Measurement function	Scale	Unit of measurement
derived	$V = \text{SpC}/\text{AoS}$	real number between 0 and 1	no unit
	If V >= 2 then **FoC = 0**		
	If 1 < V < 2 then **FoC = 2 - V**		
	If V <= 1 then **FoC= V**		

• *Measure objective*: The goal of this measure is to detect components that are (possibly) widely used and should therefore be monitored with special care because a change (or error) could affect the system more than other components that provide an AoS. On the other hand, the measure detects components that offer few or no services. Therefore our aim is to detect complex components; we believe that a component that offers twice the AoS should be checked, so we use the value of 2 which seems appropriate for this purpose.

3.2.2 Component coverage

8. *Measure name*: Component coverage (CoC)

• *Description*: This measure indicates the number of a component's services which are used by the system in relation to the total number of services which are provided by this component.

Type of measure	Measurement function	Scale	Unit of measurement
derived	UsC/SpC	real number between 0 and 1	no unit

• *Measure objective*: this indicates how many of a component's services are used by the system. The goal is to check whether the component is underused. In this case, it might be interesting to change the component for another with fewer services.

3.2.3 Component use

9. *Measure name*: Component use (CoU)

• *Description*: This indicates the ratio between the number of each component's services used with regard to the minimum number of calls to these services. Since the measure is not assessed in execution time, the real number of calls to the services cannot be calculated. A pessimistic algorithm is therefore carried out. The calls which are situated in a loop are thus counted only once.

Type of measure	Measurement function	Scale	Unit of measurement
derived	$1 - \frac{\text{UsC}}{\text{McC}}$	real number between 0 and 1	no unit

• *Measure objective*: this indicates how many of a component's services are used, thus permitting the detection of possible bottlenecks.

3.2.4 Weighted component quality

10. *Measure name*: Weighted component quality (WcQ)

• *Description*: This measure gives a weighted value of the three previous measures. All three assume an equal weight.

Type of measure	Measurement function	Scale	Unit of measurement
derived	$\frac{\text{FoC} + \text{CoC} + \text{CoU}}{3}$	real number between 0 and 1	no unit

• *Measure objective*: this measure gives us a general perception of the overall quality of each component in the

system. Therefore if the analysis of each measure of the component is not desired, this measure can be used to provide a general idea of its quality.

3.3 System utilisation measures

This section presents information related to the measures which are focused on the system quality (these are situated in Fig. 1 in the right-angle):

3.3.1 System coverage

12. *Measure name*: Coverage ratio (CoR)

- *Description*: This measure is a rate between the total number of services used in the system and the total number of services provided.

Type of measure	Measurement function	Scale	Unit of measurement
derived	$\frac{\text{UsC}}{\sum \text{Spc}}$	real number between 0 and 1	no unit

- *Measure objective*: This measure indicates the services which are used.

3.3.2 Level of use

13. *Measure name*: Level of use (LoU)

- *Description*: Rate between the total number of used services of all the components and the minimum number of calls to these services.

Type of measure	Measurement function	Scale	Unit of measurement
derived	$\frac{\text{UsS}}{\text{McS}}$	real number between 0 and 1	services/calls

- *Measure objective*: This measure shows how many of the services which are integrated into the system are used.

3.4 System integration measures

Finally, the measure related to the glue code is explained (this measure appears inside the arrow in Fig. 1)

3.4.1 Lines of glue code:

14. *Measure name*: Lines of glue code (LGC).

- *Description*: This indicates the number of lines of code needed to correctly integrate the components into the system.

Type of measure	Measurement function	Scale	Unit of measurement
base	number of lines of glue code	integer > 0	lines of code

- *Measure objective*: This measure can be used to discover whether the components which make up the system are difficult to integrate into it. In addition, it gives an idea about whether the glue code implements services which should be implemented by the components.

Once the measures have been defined, indicators and decision criteria should be defined. According to García *et al.* [7], an indicator is a measure that is derived from other measures using an analysis model (algorithm or calculation combining one or more measures with associated decision criteria), and the decision criteria are: thresholds, targets or patterns used to determine the need for action or further investigation, or to describe the level of confidence in a given result. Our approach is to define one indicator for each of the presented measures, and to define threshold values for them in order to determine the significance of the result obtained (the numerical value). The knowledge and advice of experts in CSBD have been used to define the decision criteria shown in Table 3. Specifically, the experts who worked on the definition of the decision criteria can be classified into two groups. The former is made up of the members of the research group. This group is composed of four people, all of whom have worked on CSBD. The second and the most important group is made up of 12 CSBD developers. These developers are workers in the aforementioned company which develops architectonical software components. However, the name of this company cannot be revealed owing to questions of confidentiality.

4 Visual metaphor

In this section, a general visual metaphor with which to visualise component-based software systems quality is presented. After analysing and comparing all the metaphors presented in Section 2.2, we have chosen the landscape metaphor as a starting point since:

- It permits the representation of three attributes which can be amplified to many more depending on the user's necessities.

- It permits the representation of a large quantity of information without obtaining worse legibility.

- The minimum size of the basic element is sufficiently large for it to be easily understandable to the user.

Table 3 Decision criteria for each measure

Indicator	Decision criteria		
	Acceptable	Marginal	Non-acceptable
I-FoC	[0.4–0.6)	[0.2–0.4)–[0.6–0.8)	[0.0–0.2)–[0.8–1.0]
I-CoC	[0.6–1.0]	[0.4–0.6)	[0.0–0.4)
I-CoU	[0.0–0.4)	[0.4–0.6)	[0.6–1.0]
I-WcQ	[0.4–0.6)	[0.2–0.4)–[0.6–0.8)	[0.0–0.2)–[0.8–1.0]
I-CoR	[0.6–1.0]	[0.4–0.6)	[0.0–0.4)
I-LoU	[0.0–0.4)	[0.4–0.6)	[0.6–1.0]
I-LGC	[0–10 000)	[10 000–15 000)	[15 000–∞)

• The limiting factor – occlusion – can be solved by moving the user's point of view.

• The minimum screen resolution allowed (1280 × 1024) is higher than the basic resolution (1024 × 768).

• Time to representation depends on the amount of information represented on the screen.

In our adaptation of this metaphor, component-based software system quality is represented as a city which is made up of a set of buildings (cube or rectangular prism). The city represents the whole system, whereas each building is a component of the system. The quality characteristics related to the component-based software system correspond with certain attributes of both the city and the buildings.

Although this metaphor could encode more, we have defined how to represent up to seven quality characteristics or sub-characteristics (as shown in Fig. 2) in order to take advantage of human visual perception:

• *Attributes of city*: meteorological aspects (a sun, a cloud or a storm cloud), texture of the ground (dry ground, a clear floor or grass) and neighbourhoods (lines which join groups of buildings).

• *Attributes of buildings*: height (higher or lower, depending on good or bad measure results, respectively), colour (blue to red), orientation (0° to 180°) and a television aerial (large or small, depending on good or bad measure values).

As will be observed, a set of components (buildings) are brought together to satisfy a set of requirements and thus form functional units which are more complex than a simple component (a set of grouped buildings forms a neighbourhood). Moreover, these functional units are grouped to form a system based on components (a city). The correspondence which is produced with the adapted metaphor is therefore simple and complete, that is, all the elements in the system correspond directly with the metaphor. Its simplicity lies in the common concepts of a city, a neighbourhood or a building – metaphors which are easy for everyone to associate with.

Once we have defined the metaphor's characteristics, the following step is to establish the relationship between the measures and the visual metaphor:

1. With regard to the buildings:

• *Height*: This attribute codifies the I-FoC indicator. Therefore if the component provides a good number of services with regard to the system (the measure value is close to 1) then the building will be high. In the contrary case the building will be low. Logically, the higher the buildings, the better the result.

• *Colour*: This codifies the I-CoC indicator, in such a way that if the component has a good coverage with regard to the system (the measure acquires a high value) then the colour of the building will be a very pure blue. In the

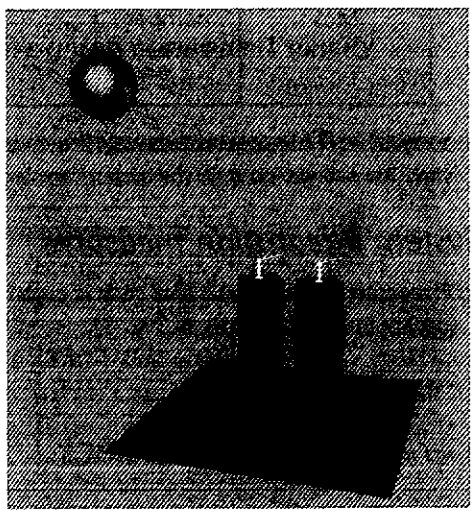


Figure 2 *Example of our visual metaphor*

contrary case, the colour of the building will be a very pure red.

• *Angle*: This attribute codifies the I-CoU indicator. The angle of the building varies between 0 and 90°. Therefore if the component is highly used in the system (the measure value is high) then the angle will be very small. In the contrary case, the angle of the building will be very high. In an ideal case the buildings will have the least possible angle.

• *Television aerial*: This attribute codifies the I-WcQ indicator. If the component is of a good quality (the measure result is close to 1) there will be three rods on the television aerial. In the contrary case, there will be only one.

2. With regard to the city:

• *The earth's surface*: This attribute represents the I-CoR indicator. If the level of CoR is high (the measure value is high), then the earth's surface will be green. However, if the level of CoR is low (the measure value is low), then the earth's surface will be brown.

• *Sky*: This attribute codifies the I-LoU indicator. The values of this attribute are: sunny, cloudy and stormy. A good LoU (a high value of the measure) implies good weather (sunny), whereas a bad use implies bad weather (stormy).

3. With regard to the neighbourhoods:

• *Separator*: This attribute represents the I-LGC indicator. Therefore if there are a lot of glue code lines, the neighbourhood separators will be thick. In the contrary case, the neighbourhood separators will be very thin.

5 ViCasBo: a component-based system quality visualisation tool

After adapting the visual metaphor and associating a measure to each metaphor element, we decided to develop a tool with which to automate the calculation and visualisation of the system quality. This tool was developed by using the platform .Net Framework 2.0 and the C#.NET programming language. The graphic engine used was DirectX. In addition, the Blender tool was used to design the 3D elements (the sun, clouds, lightning). Finally, we should emphasise that CStools was used to implement the different parsers used to extract information concerning the software component-based systems.

It should be noted that the main objective of our tool is to assess the measures which have been defined and to show them by using the visual metaphor.

Therefore the tool takes the path in which the source code of the component-based software system is located and looks for every code file through the directory hierarchy. Next, the application parses these files by calculating the values of the measures. Finally, the tool shows these measure values graphically. The visual representation of the component-based system quality allows the user to discover the quality at a glance. The tool's main screen is shown in Fig. 3.

Two actions can be carried out by our tool, as is shown in Fig. 3: (1) analyse and (2) evolution.

1. *Analysis of the system*: this option allows us to assess and graphically represent the quality of a component-based software system. The tool also can suggest corrective actions through which to improve the quality of the system. These actions are based on those measures whose values are under the desirable level, and they attempt to improve these values. Moreover, the tool presents a legend of each quality aspect in order to improve the interpretability of the metaphor (see right-hand side of Fig. 4).

2. *Evolution of system versions*: this option permits the selection of two different versions of a system and shows the quality evolution through an animation.

It should be noted that the user can move the visual representation as s/he wishes, as it is possible to zoom the image in or out and to rotate it by 360° in order to improve the visualisation.

6 Using ViCasBo tool: a case study

This section presents the result of using the tool to determine the quality of a component-based software system. This system is a real system and is owned by the company with which we are working. We are therefore unable to provide more information owing to questions of confidentiality. Suffice it to say that the main objective of the component-based software system is to make calculations for steel construction. The system is made up of ten components and there are

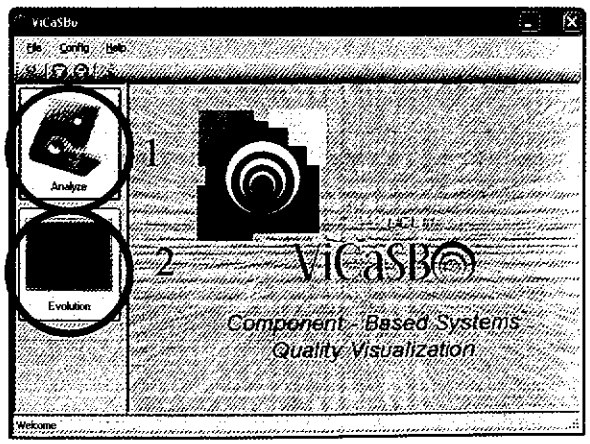


Figure 3 *Main screen of the tool*

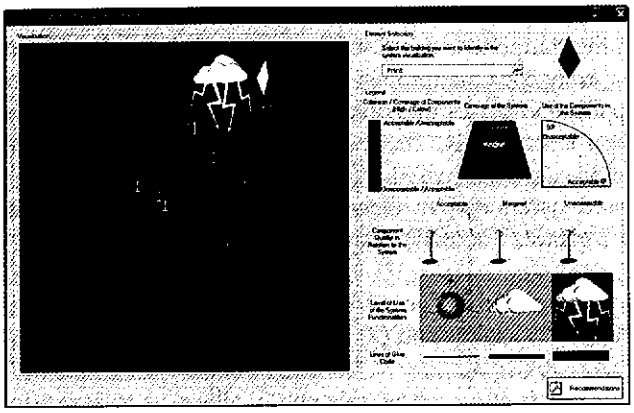


Figure 4 *Representation of the indicators for the system*

three different neighbourhoods. The neighbourhoods are obtained from the measures and the code of the system. The result obtained using our tool is shown in Fig. 4.

Some indicators have obtained values which are not desired (rated as unacceptable by the tool). The tool proposes a set of corrective actions (or recommendations) through which to improve the system quality. The corrective actions are proposed for each of the software components which are integrated into the system. One example of the corrective actions given by the tool for this system (specifically for the software component named 'one') is: the CoC must be improved, and in order to do this the system should use more of the services offered by the component.

Once the corrective actions proposed by the tool had been analysed by the developers, some of them were implemented. The system was then analysed again. The improved system is considered to be a new version, and the system quality results of this new version are shown in Fig. 5.

As Fig. 5 shows, the system quality is now better than in the previous case (Fig. 4). Specifically, the values of the two indicators which are related to the quality of the whole system have been improved. Note that in Fig. 5 the colour

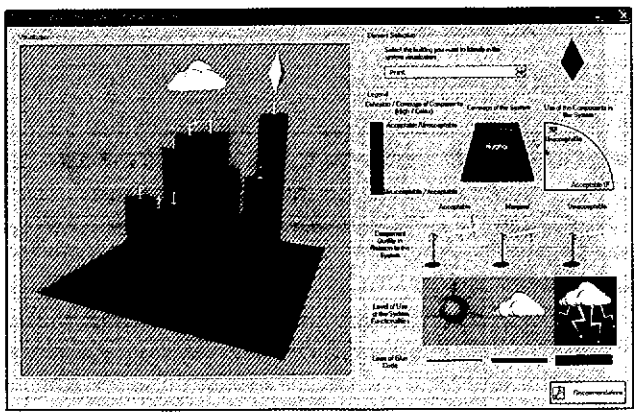


Figure 5 *Representation of the indicators for the system v.2.0*

of the earth's surface has changed. In Fig. 4 the colour was formerly brown and it is now khaki, that is, the I-CoR indicator has exchanged its unacceptable value for a marginal value. Moreover, the sky has changed. In Fig. 4 the weather was stormy whereas in Fig. 5 it is cloudy. This means that the level of the I-LoU indicator has gone from being unacceptable to being marginal.

In addition, as we have previously mentioned, the tool allows us to see the evolution between two versions of a same system thorough an animation in which the evolution of the values is shown. However, due to the dynamic nature of this action, we are unable to present an example.

Finally, it is important to emphasise that the tool allows us to personalise the criteria decisions which were presented in Table 3.

7 Conclusions and future work

The aim of this paper is to attempt to assist in the question of component-based software system quality. Our work seeks to improve the interpretation of component-based software system quality by visualising a set of indicators that affect the quality of the system. In this paper, we present a tool which graphically represents the quality of a component-based software system. To do this, the opinions of software component-oriented systems developers were considered in order to propose a set of indicators which can be automatically assessed. In addition, a visual metaphor has been selected and adapted to the context of software component-oriented systems. The visual metaphor is a city which represents the whole system and is made up of different buildings (the components which are integrated into the system). Finally, it is possible to identify neighbourhoods which represent a set of related components.

The tool presented is able to propose corrective actions through which to improve the quality of the system. The developer can carry out all or some of these corrective actions and analyse the quality of the system again. It is also possible to dynamically view the evolution between two versions of a system.

As an example, the tool has been used to analyse a component-based software system. In addition, some of the corrective actions which are proposed by the tool have been carried out. Finally, the second version of the system has been analysed, and better results were obtained than in the previous case.

As future work, we wish to work on the validation of the proposed indicators in order to assure their correctness, not only in definition, but also in the decision criteria defined since, although we believe that they are correct as a result of their having been defined through the use of expert opinion, some empirical work must be carried out to confirm this thesis.

A further line of future work will be to define a visual metaphor with which to represent the quality of the component-based software system along with the quality of each component. The tool will, therefore, first be able to visualise the quality of the whole system. Upon pressing or selecting one component the tool will then change the visual metaphor and show the quality of this component (this quality will be independent of the system into which it is integrated).

This new approach will allow us to prove whether the selected metaphor is the most appropriate or whether it is preferable to exchange it for another.

8 Acknowledgments

This work is part of the INCOME project (PET2006-0682-01) supported by the Spanish Ministerio de Educación y Ciencia, by the IVISCUS project (PAC08-0024-5991) supported by Consejería de Educación y Ciencia (JCCM) and by VIMECUS (TC20080556) supported by the University of Castilla-La Mancha.

9 References

[1] ABOWD G., BASS L., CLEMENTS P., KAZMAN R., NORTHROP L., ZAREMSKI A.: 'Recommended best industrial practice for software architecture evaluation'. Technical report CMU/SEI-96-TR-025 and ESC-TR-96-025, 1997, URL: http://www2.umassd.edu/SWPI/sei/tr025.96.pdf

[2] SZYPERSKI C.: 'Component software. Beyond object-oriented programming' (Addison-Wesley and ACM Press, 2002)

[3] CECHICH A., PIATTINI M., VALLECILLO A.: 'Assessing component-based systems', in CECHICH A., PIATTINI M., VALLECILLO A. (EDS.): 'Component-based software quality: methods and techniques' (*LNCS*, **2693**), pp. 1–20 (Spinger, 2003)

[4] YUNNI X., HANPIN W., CHUNXIANG X: 'Stochastic modeling and quality evaluation of component-based software systems'. Proc. Sixth Int. Conf. on Quality Software (QSIC'06), Beijing, China, October 2006, pp. 377–384

[5] GERSHON N.D.: 'From perception to visualization', in ROSENBLUM L., EARNSHAW R.A., ENCARNACAO J., HAGEN H., KAUFMAN A., KLIMENKO S., NIELSON G., POST F., THALMANN D. (EDS.): 'Scientific visualization: advances and challenges', pp. 129–139 (Academic Press, 1994)

[6] SEDIGH-ALI S., GHAFOOR A., PAUL R.A: 'Metrics-guided quality management for component-based software systems'. Proc. 25th Annual Int. Computer Software and Applications Conf. (COMPSAC'01), Chicago, Illinois, 2001, pp. 303–310

[7] GARCÍA F., BERTOA M.F., CALERO C., VALLECILLO A., RUÍZ F., GENERO M.: 'Towards a consistent terminology for software measurement', *Inf. Softw. Technol.*, 2006, **48**, (8), pp. 631–644

[8] GOULÃO M., BRITO E ABREU F: 'Software components evaluation: an overview'. Proc. Fifth Conf. on Associaçao Portuguesa de Sistemas de Informaçao (CAPSI), Lisbon, 2004, URL: http://www-ctp.di.fct.unl.pt/QUASAR/Resources/Papers/2004/goulaoCAPSI2004Final.pdf

[9] DIEHL S.: 'Software visualization. Visualizing the structure, behaviour, and evolution of software' (Springer, 2007)

[10] EICK S.G., SCHUSTER P., MOCKUS A., GRAVES T.L., KARR A.F.: 'Visualizing software changes', *IEEE Trans. Softw. Eng.*, 2000, **28**, (4), pp. 396–412

[11] EICK S.G., KARR A.F.: 'Visual scalability'. Technical report, 2000, National Institute of Statistical Sciences, URL: http://www.niss.org/technicalreports/tr106.pdf

[12] LANGELIER G., SAHARAOUI H., POULIN P.: 'Visualization-based analysis of quality for large-scale software systems'. Proc. 20th IEEE/ACM Int. Conf. on Automated Software Engineering, California, USA, November 2005, pp. 214–223

[13] MARCUS A., FENG L., MALETIC J.I.: '3D representations for software visualization'. Proc. ACM Symp. on Software Visualization. IEEE Software Society, San Diego, California, 2003, pp. 27–37

[14] LANZA M., DUCASSE S.: 'Polymetric views – a lightweight visual approach to reverse engineering', *IEEE Trans. Softw. Eng.*, 2003, **29**, (9), pp. 782–795

[15] CROSS J.H.: 'Grasp: graphical representations of algorithms, structures, and processes', http://www.eng.auburn.edu/grasp, accessed July 2008

[16] AbsInt Angewandte Informatik GmbH: 'Call graph visualization and stack usage analysis', http://www.aisee.com/aicall/, accessed July 2008

[17] Goose Homepage: http://esche.fzi.de/PROSTextern/software/goose/index.html, accessed July 2008

[18] Arisa: http://www.arisa.se/vizz_analyzer.php, accessed July 2008

[19] MEHNER K.: 'JaVis: A UML-based visualization and debugging environment for concurrent Java programs', in DIEHL S. (EDS.): 'Software visualization: international seminar' (*LNCS*, **2269**), pp. 163–175 (Springer, Verlag, 2002)

[20] STASKO J.T.: 'TANGO: a framework and system for algorithm animation', *Computer*, 1990, **23**, (9), pp. 27–39

[21] PINZGER M., GALL H., FISCHER M., LANZA M.: 'Visualizing multiple evolution metrics'. Proc. 2005 ACM Symp. on Software Visualization, St. Louis, Missouri, May 2005, pp. 67–75

[22] MCGARRY J., CARD D., JONES C., LAYMAN B., CLARK E., DEAN J., HALL F.: 'Practical software measurement' (Addison-Wesley, 2002)

[23] SIMÃO R.P., BELCHIOR A.: 'Quality characteristics for software components: Hierarchy and quality guides', in CECHIC A., PIATTINI M., VALLECILLO A. (EDS.): 'Component-based software quality', pp. 184–206, (Springer Verlag, 2003)

[24] BERTOA M.F., VALLECILLO A.: 'Quality attributes for COTS components'. Proc. Sixth Int. Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE'2002), Málaga, 2002, pp. 54–66

[25] SEDIGH-ALI S., GHAFOOR A., PAUL R.A.: 'Software engineering metrics for COTS-based systems', *IEEE Comput.*, 2001, **34**, (5), pp. 44–50

[26] GILL N.S., GROVER P.S.: 'Component-based measurement: few useful guidelines', *ACM SIGSOFT Softw. Eng. Notes*, 2001, **28**, (6), pp. 1–6

[27] DUMKE R., SCHMIETENDORF A.: 'Possibilities of the description and evaluation of software components', *ACM SIGSOFT Softw. Eng. Notes*, 2000, **5**, (1), pp. 13–26

[28] BOXALL M.A.S., ARABAN S.: 'Interface metrics for reusability analysis of components'. Proc. 2004 Australian Software Engineering Conf. (ASWEC'04), Melbourne, Australia, April 2004, pp. 40–51

[29] WASHIZAKI H., YAMAMOTO H., FUKAZAWA Y.: 'A metrics suite for measuring reusability of software components'. Proc. Ninth Int. Software Metrics Symp., Sydney, Australia, September 2003, pp. 211–223