Available online at www.sciencedirect.com

ScienceDirect

Information and Software Technology 49 (2007) 108–121

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.com/locate/infsof

Early detection of COTS component functional suitability

Alejandra Cechich [a,*], Mario Piattini [b,1]

[a] Departamento de Ciencias de la Computación, Universidad Nacional del Comahue, Buenos Aires 1400 (8300) Neuquén, Argentina
[b] Escuela Superior de Informática, Universidad de Castilla-La Mancha, Paseo de la Universidad 4, (13701) Ciudad Real, Spain

Received 11 March 2005; received in revised form 13 March 2006; accepted 22 March 2006
Available online 13 June 2006

Abstract

The adoption of COTS-based development brings with it many challenges about the identification and finding of candidate components for reuse. Particularly, the first stage in the identification of COTS candidates is commonly carried out by dealing with unstructured information on the Web, which makes the evaluation process highly costly when applying complex evaluation criteria. To facilitate this process, our proposal introduces an early measurement procedure for suitability of COTS candidates. Considering that filtering is about a first-stage selection, functionality evaluation might drive the analysis, proceeding with the evaluation of other properties only on the pre-selected candidates. In this way, a few candidates are fully evaluated making in principle the whole process more cost-effective. In this paper, we illustrate how functional measures at an initial state are calculated for an E-payment case study.
© 2006 Elsevier B.V. All rights reserved.

Keywords: COTS (Commercial Off-The-Shelf) components; Component-based system development; COTS component selection; Quality attributes; Six-Sigma

1. Introduction

COTS-based development changes the focus of software engineering from one of traditional system specification and construction to one requiring simultaneous consideration of the system context (system characteristics such as requirements, cost, schedule, operating and support environments), capabilities of products in the marketplace, and viable architectures and designs. The impact of this fundamental change is profound. Not only must engineering activities such as requirements specification change to support these simultaneous consideration, but so must acquisition processes and contracting strategies.

It is clear that COTS-Based Software Engineering affects software quality in several ways, ranging from introducing new methods for selecting components to defining a wide scope of testing principles and measurements [1]. Today, software quality staff must rethink the way software is assessed, including all life-cycle phases – from requirements to evolution.

When selecting COTS components, there is usually a quick first effort intended to determine very rapidly which products are unsuitable in the current context. This process can be initiated as soon as there is at least one relevant alternative to consider. This initial *filtering* may be carried out based on different information to be collected, including vendor capability, legal aspects on the use of the component, offered functionality, etc.

One of the reasons of not using a formal selection process for filtering lies in the key motivation of using COTS components in the system [2,3]. Usually, it is to increase time-to-market and reduce cost, therefore composers do not spend much effort on evaluating all candidate components by using a formal selection process because it does not look cost-effective.

[*] Corresponding author. Tel.: +54 299 4490312; fax: +54 299 4490313.
E-mail addresses: acechich@uncoma.edu.ar (A. Cechich), Mario.Piattini@uclm.es (M. Piattini).
[1] Tel.: +34 926 295300; fax: +34 926 295354.

0950-5849/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2006.03.008

The point is that many methods include complex criteria and qualifying thresholds for filtering. For example, during the activity "Collect Measures" of the COTS Acquisition Process [4], data according to a measurement plan and based on a taxonomy are collected on a set of COTS software alternatives. However, considering that filtering is about a first-stage selection, functionality evaluation might drive the first analysis, proceeding with the evaluation of other properties only on the pre-selected candidates.

Then, we mirror some efforts in literature [5–7], meaning that functionality is our main concern, but addressing how functional suitability might be quantified at early stages of the selection process. Differently from those proposals, our metrics [8] are defined considering that only a high-level description of the COTS components is available, without further details on data and control execution. This assumption is supported by fundamentals of the filtering process, and by the characteristics of information provided by most component suppliers [9,10].

Our process [11,12] aims at reducing the number of candidates by selecting some of them very quickly based on a brief review on key functional issues. These aspects, along with some cost and market issues, may help introduce a more formal review later on. In this case, only a few candidates are fully evaluated making in principle the whole process more cost-effective.

We should note that although functionality guides our process, the problem of ensuring component and component composition quality is exacerbated by the multiplicity of potential problems with COTS components. Whether you have built your system using COTS components from many vendors, or a single vendor has provided you with an integrated solution, many of the risks associated with system management and operation are not in your direct control [13].

Therefore, a quality assessment team usually launches its efforts by identifying major organizational processes and their customers. At this point, the objective is to prepare a list of activities related top COTS component selection, where spiral approaches define a series of iterations to mitigate risk while addressing the most critical functions [14,15]. Similarly, our measurement process is immerse into a broader iterative approach supported on Six-Sigma [16–19] precepts. Our approach defines five-phases for the pre-selection process; however in this paper we focus on the second phase only: measurement of functional suitability. We refer the reader to [11,12] for a description of the whole process and other architectural measures calculated during filtering.

In this paper, we introduce the functional measurement of COTS components as follows. Firstly, Section 2 presents some related works. Then, Section 3 introduces our suite of metrics; and Section 4 shows how those metrics are applied to a case study. Section 5 addresses lessons learned; and finally, we draw conclusions as well as present future work in this domain.

2. Related works

So far, many methods and techniques have emerged to address the evaluation and selection of COTS candidates [1]. In spite of that fact that an exhaustive review is out of the scope of this paper, we would like to introduce those proposals that we consider closer to ours.

A first related work by Bianchi et al. [6] defines the quality characteristic "adequateness" of a COTS product by considering the application domain, the functionality provided by the COTS product, and its effective usage within the Component-Based System (CBS). Authors have introduced two metrics – "Functional Coverage" and "Compliance" – to refer to the concept of functionality as broadly as possible. The former expresses the percentage capability of each COTS product to support the functionality required by the CBS; the latter expresses the effective usage of each COTS product within CBS. For example, the value of the Functional Coverage metric for the *i*th COTS product is calculated from the number of functionality provided to the CBS by the *i*th COTS product, and the total number of functionality provided by all the COTS products to the CBS. The rationale behind our proposal is similar to those considerations; however, they do not address the problem of calculating "the number of functionality" as required by the metrics. Our approach builds upon the functional view of this proposal, but details calculations by abstractly defining functionality as well as data required for measurement.

The proposal by Albert et al. [15], called *Evolutionary Process for Integrating COTS-Based Systems* (EPIC), builds upon of the elements of the Rational Unified Process [20], and disciplined spiral engineering practice [14]. Particularly interesting to our work is the "Inception Phase", which establishes a common understanding among stakeholders of what the solution will do. It ends when it is demonstrated that one or more candidate solutions can be integrated into the organization's architecture. This phase accumulates information to identify measurable criteria that correspond to preliminary expectations; however, the lack of particular measurement procedures – and metrics – makes this process a candidate to improvement. Our approach for COTS component filtering is build upon the notion of improvement, but explicitly including measurement during the process. We immerse our techniques into a Six-Sigma-based process, aiming at defining what is relevant for filtering, measuring it, and improving the filtering practices. We limit the scope of this paper to only one calculation during this process, that is functional suitability measurement.

The *Function Fit Analysis* [7] consists of five primary steps. Firstly, the *Requirements Function Point Analysis* is used to identify and document what functionality is necessary to meet the customer needs. The analysis focuses on quantifying user requirements and the result is a development function point count and a detailed listing of the functions necessary to meet the requirements. Secondly, the *COTS Functional Evaluation* step involves reviewing the various COTS candidates and comparing their functionality to the requirements previously documented. Thirdly, the *Functional Fit Analysis* applies a percentage of "fit" between the requirements and the COTS product. In Function Fit Analysis, "fit" is defined as the amount of out-of-the-box functionality that can be used without any modifications. Using this definition, a comparison of the requirements function point count to the COTS function point count results in calculating the "fit" percentage of the COTS. However, differently from our proposal, what functionality means is not explicitly addressed.

PORE (*Procurement-Oriented Requirement Engineering*) [21] selects products by rejection, i.e., products that do not meet core customer requirements are selectively and iteratively rejected and removed from the candidate list. This process shows that increasing the number and detail of requirement statements will decrease the number of COTS candidates. For requirements acquisition, PORE divides the process into stages and provides three templates for three key stages of the process [22]. Particularly, the first template is used during early stages of COTS component selection, when the evaluation team deals with Web site information, technical documents, market analyses, etc., which are characterized by the lack of detailed data.

One of the problems of the PORE method is that the iterative process of requirements acquisition and product evaluation/selection is very complex. At any point, a large number of possible situations can arise, or a large number of processes and techniques to use in a single situation can be recommended. To handle this scale of complexity, a prototype tool known as PORE Process Advisor has been developed. The main components of the tool are a process engine which analyses the current set of goals to be achieved, model properties (inferred by the situation inference engine) and instructions from the requirements engineering team to recommend process advise. Complexity of filtering is precisely the point to discuss here. Although PORE provides a tool for supporting the search, complexity of the procedure itself may not look cost-effective.

Finally, our measures on functional suitability might provide a more precise indicator when calculating the *maintenance equilibrium value* as introduced by Abts in [5]. In this proposal, the number of components in the solution should be minimized and the contribution of functionality should be maximized to satisfy the CBS Functional Density Rule of Thumb: "*Maximize the amount of functionality in your system provided by COTS components but using as few COTS components as possible*" [5]. However, there is no further detail on what "amount of functionality" is.

3. Measuring COTS component functional suitability

To ensure that decisions are fact based, it is important that definitions used during the process be reinforced. Although it is always important to understand the current process and the problem to be solved, the importance is greater when filtering COTS components because requirements and services must be balanced as part of a negotiation procedure.

Firstly, a COTS component's required functionality should be expressed to define search goals and criteria for evaluation. To do so, we have adapted the model in [23], which explores the evaluation of components using a specification-based testing strategy, and proposes a semantics distance measure that might be used as the basis for evaluating a component from a set of candidates. This model supposes that there is an architectural definition of a system, whose behavior has been depicted by scenarios or using an architecture description language (ADL). Precisely, the possibility of combining more formal descriptions of functional requirements and scenarios, motivated us to choose this approach as a starting point. This combination allows us to split complexity from a functional point of view (by using scenarios) and reason about the needs of information processing in terms of provided and required data. Let us briefly discuss the model.

According to Alexander [23], a system can be extended or instantiated through the use of some component type. Though several instantiations might occur, an assumption is made about what characteristics the actual components must possess from the architecture's perspective. Thus, the specification of the architecture A (S_A) defines a specification S_C for the abstract component type C (i.e., $S_A \Rightarrow S_C$). Any component K_i, that is a concrete instance of C, must conform to the interface and behavior specified by S_C, as shown in Fig. 1 (from [23]).

The process of composing a component K with A is an act of interface and semantic mapping. There are many techniques currently addressing interface mapping, hence in this paper, only the semantic mapping will be addressed. We focus on incompatibilities derived from behavioral differences between the specification of a component K_i (S_{K_i}) and the specification S_C. Another necessary condition for using K (or a combination of K_i) to satisfy S_C is that the input and output domains of K include some of those specified by S_C. An additional necessary condition is that K

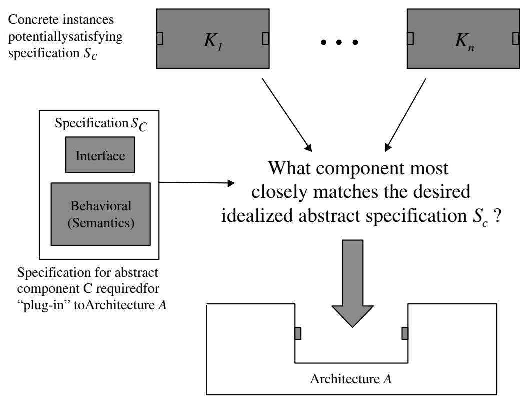
Concrete instances potentiallysatisfying specification S_c
K_1
K_n
Specification S_C
Interface
Behavioral (Semantics)
What component most closely matches the desired idealized abstract specification S_c ?
Specification for abstract component C requiredfor "plug-in" toArchitecture A
Architecture A

Fig. 1. Instantiation of an abstract component specification.

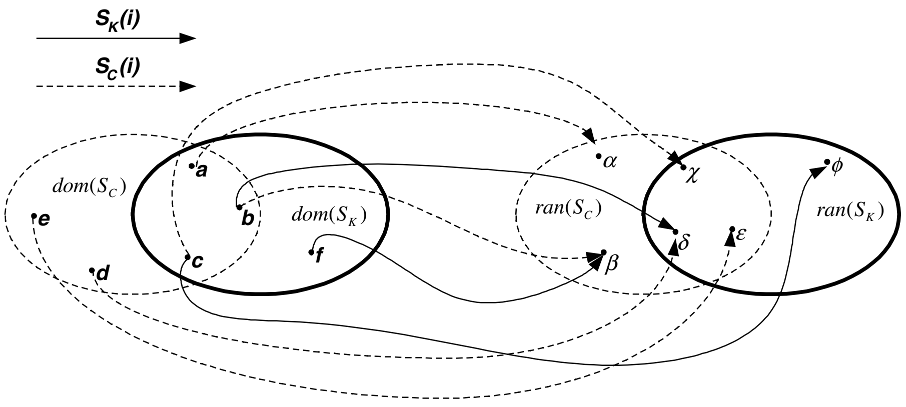

Fig. 2. Functional mappings of S_C and S_K.

provides at least some functional mappings between the domains as specified by S_C.

The model, as illustrated in Fig. 2, helps us to identify inconsistencies and reason about required and provided services abstractly. Precisely, this abstract view is what we need to deal with coarse-grained information at the filtering level. A typical situation for inconsistency in the functional mappings between S_K and S_C is illustrated in Fig. 2 (from [23]), where the dashed lines indicate mappings with respect to S_C, and the solid lines are mappings with respect to S_K. Note that the input (dom) and output (ran) domains of S_K and S_C are not equal. Also, the domain of S_C is not included in the domain of S_K, and vice versa for the ranges.

3.1. The measurement phase

Let us present our case. Suppose that before starting the evaluation procedure, a committed specification of the component (S_C) is defined as a reference to be used when comparing candidates [24]. Once committed goals have been achieved, we may proceed with the filtering process by applying different measures on COTS component candidates.

For the measure definitions, we assume a conceptual model with universe of scenarios $\mathscr{S}$, an abstract specification of a component $\mathscr{C}$, a set of components $\mathscr{K}$ relevant to $\mathscr{C}$ called candidate solution $\mathscr{SO}$, a set of pre-selected components from $\mathscr{SO}$, called solution $\mathscr{SN}$, and a mapping component diagram $\mathscr{MCD}$ as we introduced previously (Fig. 2). In this diagram, $S_C(i)$ represents the map associated to the input value i defined in the domain of $\mathscr{SC}$. This map should provide a valid value on the output domain of $\mathscr{SC}$, i.e., there is no empty maps or invalid associations. A similar assumption is made on the mappings of $\mathscr{SK}$.

We should note that perception of a composer will be affected by many factors such as his own capability and experience in searching and using COTS components; the complexity of the particular domain application; the granularity of the component he/she is looking for; and so forth. Therefore, mappings are instantiated during a particular filtering letting composers adapt it to different levels of COTS-based development maturity. For example, let us consider a possible instantiation of the functional description case – abstractly "scenarios" of our process. A particular organization might use recommendations from the ISO/IEC 19761 standard – Software engineering – COSMIC-FFP, which introduces a functional size measurement method [25] to instantiate the definition of scenarios. The method considers that the functionality delivered by software to its users is described through the Functional User Requirements (FUR) documents. These state "what" the software must do for the users and exclude any technical or quality requirements that say "how" the software must perform. In practice, FUR sometimes exist in the form of a specific document (requirements specifications, for instance), but often they have to be derived from other software engineering artifacts. FUR can be derived from software engineering artifacts that are produced before the software exists (typically from architecture and design artifacts). Thus, the functional size of software can be measured prior to its implementation on a computer system. In other circumstances, software might be used without there being any, or with only a few, architecture or design artifacts available, and the FUR might not be documented (legacy software, for instance). In such circumstances, it is still possible to derive the software FUR from the artifacts installed on the computer system even after it has been implemented.

Finally, let us briefly clarify the concepts associated to $\mathscr{SO}$ and $\mathscr{SN}$. Candidate components, selected from different sources for evaluation, constitute the members of the set $\mathscr{SO}$. It could be the case that one of these members does not offer any functionality required by $\mathscr{C}$. Hence, an evaluator should not spend more time and effort evaluating other properties or requirements on that component, i.e., the component should be withdrawn from analysis. Then, the solution in which all components potentially contribute with some functionality to fulfil the requirements of $\mathscr{C}$ is called here $\mathscr{SN}$.

Table 1
Description of the Functional Suitability measures

Measure Id.	Description
Component-level	
CF_C Compatible Functionality	The number of functional mappings provided by S_K and required by S_C in the scenario $\mathscr{S}$
MF_C Missed Functionality	The number of functional mappings required by S_C in the scenario $\mathscr{S}$ and NOT provided by S_K
AF_C Added Functionality	The number of functional mappings NOT required by S_C in the scenario $\mathscr{S}$ and provided by S_K
CC_F Component Contribution	Percentage in which a component contributes to get the functionality required by S_C in the scenario $\mathscr{S}$
Solution-level	
SN_{CF} Candidate Solution	The number of components that contribute with compatible functionality to get the requirements of $\mathscr{S}_{\mathscr{C}}$ in the scenario $\mathscr{S}$
CF_S Compatible Functionality	The number of functional mappings provided by $\mathscr{S}\mathscr{N}$ and required by S_C in the scenario $\mathscr{S}$
MF_S Missed Functionality	The number of functional mappings required by S_C in the scenario $\mathscr{S}$ and NOT provided by $\mathscr{S}\mathscr{N}$
AF_S Added Functionality	The number of functional mappings NOT required by S_C in the scenario $\mathscr{S}$ and provided by $\mathscr{S}\mathscr{N}$
SC_F Solution Contribution	Percentage in which a solution contributes to get the functionality required by S_C in the scenario $\mathscr{S}$

3.1.1. Functional suitability measures

In the following definitions, we use the symbol # for the cardinality of a set. To simplify the analysis, we also assume input/output data as data flows, i.e., data that may aggregate some elemental data. It means that the particular data composition, such as iterative or alternative structures, is actually hidden.

Table 1 lists our suite of functional suitability measures. The measures have been grouped into two main groups: component-level measures and solution-level measures. The first group of metrics aims at detecting incompatibilities on a particular component $\mathscr{K}$, which is a candidate to be analyzed. However, it could be the case that we need to incorporate more than one component to satisfy the functionality required by the abstract specification $\mathscr{C}$. In this case, the second group of metrics evaluates the domain compatibility of all components that constitute the candidate solution $\mathscr{S}\mathscr{O}$, as we previously defined.

For example, CF_C measures the number of functional mappings provided by S_K and required by S_C in the scenario $\mathscr{S}$; SN_{CF} measures the number of components that contribute with compatible functionality to get the requirements of $\mathscr{S}_{\mathscr{C}}$ in the scenario $\mathscr{S}$; and so forth.

Note that CC_F is an *indirect* measure, i.e. a measure of an attribute that is derived from measures of one or more other attributes. Also, note that the CC_F metric contains all information we need during the functional suitability analysis. However, the other metrics are useful in order to identify needs of adaptation later on during our filtering process [12].

A more formal definition of the measures is shown in Table 2, where comparison between output domain values has been simplified by considering equality. A more complex treatment of output values might be similarly specified, for example, by defining a set of data flows related by set inclusion.

The metrics listed here are calculated during the first stage of our filtering process, aiming at providing a coarse indicator of suitability on analyzed components. Measuring other aspects is still a remaining issue, which might be addressed by measuring architectural features at early stages. We refer the interested reader to [11,12,26] for a detailed definition of the process and the use of architectural metrics during filtering.

Another interesting discussion will be on analyzing the representation of the input/output domain, trying to close the gap between the information provided by component vendors and the information required for evaluation, as the work in [9] remarks. Additionally, we should remark the importance of standardizing and evaluating information sources. Further discussion will be introduced in Section 5.

4. Case study – E-payment by credit card

The functional suitability measurement is studied through analyzing different kinds of document material related to the COTS candidates. It includes material on the Web collected from portals for searching components; particularly in our case, information from the component-Source organization.[2]

In choosing the case, the most important criteria is that we should maximize what we can learn through the case [27]. Eisenhardt [28] has suggested that cases should be chosen for theoretical, not statistical reasons. The case is to be selected so that it is a typical or representative of other cases. However, no matter how much effort is put into finding a typical case, it is unlikely that we can accomplish a strong representation of others. Especially with COTS components, which are developed to supply very simple services as well as to support entire domain applications.

In our case study, we want to analyze a specific case in order to learn more about other cases as well. Thus, a certain amount of typicality is needed in selecting the case. However, determining the criteria for 'typicality' may, of course, be a complex task. In this, the most important guideline is the purpose of the study, which should form the basis for determining the case selection criteria. In addition to this, other very important issues influencing the selection are factors such as time limitations and access.

[2] www.componentsource.org

Table 2
Functional Suitability measures

Measurement element	Measure Id.	Measure definition
Component $\mathcal{K}$ and component $\mathcal{C}$	CF_C	$\#\{df\colon data \mid df \in (dom\ S_C \cap dom\ S_K) \wedge S_K(df) = S_C(df)\}$
Component $\mathcal{K}$ and component $\mathcal{C}$	MF_C	$\#\{df\colon data \mid CF_C(K, C) \geqslant 1 \wedge df \in dom\ S_C \wedge (S_C(df) \neq S_K(df) \vee df \notin dom\ S_K)\}$
Component $\mathcal{K}$ and component $\mathcal{C}$	AF_C	$\#\{df\colon data \mid CF_C(K, C) \geqslant 1 \wedge df \in dom\ S_K \wedge (S_C(df) \neq S_K(df) \vee df \notin dom\ S_C)\}$
CF_C and component $\mathcal{C}$	CC_F	$\frac{CF_C}{\#(S_C)}$
Solution $\mathcal{SO}$ and component $\mathcal{C}$	SN_F	$\#\{S_K \mid (CF_C(K, C) \geqslant 1) \forall K \in SO\}$
Solution $\mathcal{SN}$ and component $\mathcal{C}$	CF_S	$\#\{df\colon data \mid df \in (dom\ S_C \cap \cup\,(dom\ S_K) \forall K \in SN) \wedge (S_K(df) = S_C(df)) \exists K \in SN)\}$
Solution $\mathcal{SN}$ and component $\mathcal{C}$	MF_S	$\#\{df\colon data \mid df \in (dom\ S_C \setminus \cup\,(dom\ S_K) \forall K \in SN) \vee (df \in dom\ S_C \cap \cup\,(dom\ S_K) \forall K \in SN \wedge (S_C(df) \neq S_K(df)) \forall K \in SN)\}$
Solution $\mathcal{SN}$ and component $\mathcal{C}$	AF_S	$\#\{df\colon data \mid df \in \cup\,(dom\ S_K) \forall K \in SN \setminus dom\ S_C \wedge (S_K(df)) \exists K \in SN)\}$
CF_S and component $\mathcal{C}$	SC_F	$\frac{CF_S}{\#(S_C)}$

The purpose of this research is to calculate an early measure of COTS functional suitability. In this case, a typical situation and an appropriate case is one where:

(1) the *domain application* is broad enough to detect different COTS candidates on the market.
(2) *knowledge* needed to detect candidates is not specific or requires in-depth training on the particular domain.
(3) the *component granularity* is middle-grained, which means that a component supplies several functions but not so many that a composer would not be able to handle their complexity.

On the basis of the discussed criteria for finding the case, we will focus on E-payment by credit card. Electronic payment accomplishes the criteria afore mentioned since there are more than enough COTS candidates on the market; knowledge is broad enough to be easily understood; and functionality is split into authorization and capture, which in turn imply the existence of several functions.

Table 3 shows the contextual use case, using a basic template for use cases matching the document "Structuring Use Cases with Goals".[3] The template has the following sections: name (which is the goal), goal in context, scope, level, trigger, pre- and post-conditions, main course, extensions, sub-variations, and other characteristic data for the use case.

From action 3 in Table 3, a service provider intermediates in credit card transaction processing. It implies that the service provider validates credit card numbers and expiration dates, obtains authorization from the credit card issuers and issues confirmation numbers to taxpayers at the end of the payment transaction.

For the sake of this example, we suppose that the required functionality will be supplied by COTS components in a marketplace. So, the next step is to define mappings for the filtering process.

Generally speaking, *Authorization* and *Capture* are the two main stages in the processing of a card payment over the Internet. Authorization is the process of checking the customer's credit card. If the request is accepted the customer's card limit is reduced temporarily by the amount of the transaction. Capture is when the card is actually debited. This may take place simultaneously with the authorization request.

By using a traditional technique for deriving goals from scenarios, we might produce a simplified abstract specification of the input and output domains of S_C as follows. Combination of and relationships among different scenarios might be similarly treated by following the guidelines of derivation introduced in [29].

- Input domain:
  (AID) Auth_IData{#Card, Cardholder_Name, Exp_Date, Bank_Acc, Amount};
  (CID) Capture_IData {Bank_Acc, Amount}.
- Output domain:
  (AOD) Auth_OData{ok_Auth};
  (COD) Capture_OData{ok_capture, DB_update}.
- Mapping: $\{AID \mapsto AOD;\ CID \mapsto COD\}$.

The component source for this study includes all COTS components catalogued as members of the "Credit Card Authorization" group by the ComponentSource organization. The following sections introduce some examples of our analysis, which was developed during October, 2004 [30]. For brevity reasons, we show how measures were calculated for only five components, considering that a similar treatment was given to all components in the group. Table 5 lists all components surveyed and analyzed; however note that some of them, such as CCValidate, are not longer available.

4.1. The AcceptOnline case

AcceptOnline is a COM object that provides credit card processing functionality for developers. It connects to the credit card processor server through the Internet via SSL protocol which guarantees secure data transfer. To minimize fraud transactions Address Verification Service (AVS) is supported. AcceptOnline COM object provides the ability to accept credit or debit cards from Web-site or Internet-enabled applications. Credit card transactions

[3] Available at http://alistair.cockburn.us/spstab3s/uctempla.htm

Table 3
Contextual use case

USE CASE	Pay taxes by credit card	
Goal in context	Individual taxpayers can e-file and make a payment by credit card	
Scope & Level	Department of the Treasury, USA Primary Task	
Preconditions	We know taxpayers and the amount of tax owed	
Success End Condition	Credit card approval: the taxpayer's account has money for the payment	
Failed End Condition	Taxes are not paid. The taxpayer has not spent the money	
Primary, Secondary Actors	Individual taxpayer, any agent acting for the taxpayer. Department of the Treasury, Payment service provider	
Trigger	Payment request comes in	
Description	Step	Action
	1	Taxpayer calls in with a payment request
	2	Department of the Treasury captures primary SSN, secondary SSN, credit card number, credit card expiration date, etc
	3	Service provider intermediates in credit card transaction processing
	4	Department of the Treasury gives taxpayer information about convenience fees, interest, service provider fee, etc
	5	Taxpayer accept payment
	6	Service provider issues a confirmation number
EXTENSIONS		Branching action
	1a	Rejected e-file and e-pay 1a1. Renegotiate payment
SUB-VARIATIONS		
	1	Taxpayer may use Phone in, Web page form

are processed through the secure Internet connection (with aid of the SSL protocol). For a credit card processor, the ECHO[4] is used. All major credit cards are supported. To accept credit cards from a customer we should get the merchant account from credit card processor.

Properties of AcceptOnline are grouped into the following classes: merchant fields, transaction fields, and response fields. From those classes, we identify:

- transaction_type: this field identifies the type of transaction being submitted. Valid transaction types are:
  – "CK" (System check),
  – "AD" (Address Verification),
  – "AS"(Authorization),
  – "ES" (Authorization and Deposit),
  – "EV" (Authorization and Deposit with Address Verification),
  – "AV" (Authorization with Address Verification),
  – "DS" (Deposit), and
  – "CR" (Credit).
- cc_number: the credit card number to which this transaction will be charged,
- cc_exp_month and cc_exp_year: the numeric month (01–12) and the year (formatted as either YY or CCYY) in which this credit card expires,
- billing_phone: the shopper's telephone number,
- grand_total: the total amount of the transaction,
- merchant_email: this is the Email address of the merchant,
- order_type: this field determines which fields are used to validate the merchant and/or hosting merchant,
- transactionStatus: Transaction Status. Valid values are: G – Approved, D – Declined, C – Cancelled, T – Timeout waiting for host response, R – Received.

Methods of AcceptOnline are specified in terms of their main focus and required input. Particularly, the *SendPacket* method is used to send the transaction information to the ECHOOnline server, and required properties should be filled as shown in Table 4 (requirements for CR are partially listed). Note that order_type, transaction_type, and merchant_email should be always filled before calling the method SendPacket of AcceptOnline, so they are not listed on Table 4.

From the AcceptOnline (AOnline) description above, we might derive the following mappings related to our authorization (AS) and capture (DS) required functionality:

- Input domain (from Table 4)
  – (AOnline.ASI) {billing_phone, cc_number, cc_exp_month, cc_exp_year, counter, debug, grand_total, merchant_email};
  – (AOnline.DSI) {authorization, cc_number, cc_exp_month, cc_exp_year, counter, debug, grand_total, merchant_email}.

[4] www.echo-inc.com

Table 4
Required fields by transaction type of AcceptOnline

Field	CK	AD	AS	ES	EV	AV	DS	CR
authorization							Y	
billing_address1		Y			Y	Y		
billing_address2								
billing_zip		Y			Y	Y		
billing_phone		Y	Y		Y	Y		
cc_number		Y	Y	Y	Y	Y	Y	Y
cc_exp_month		Y	Y	Y	Y	Y	Y	Y
cc_exp_year		Y	Y	Y	Y	Y	Y	Y
counter			Y	Y	Y	Y	Y	Y
debug		Y	Y	Y	Y	Y	Y	Y
grand_total			Y	Y	Y	Y	Y	Y
merchant_email		Y	Y	Y	Y	Y	Y	Y
order_number							Y	Y
...								...

- Output domain
  - (AOnline.ASO) {TransactionStatus};
  - (AOnline.DSO) {TransactionStatus}.
- Mapping
  - $\{AOnline.ASI \mapsto AOnline.ASO\}$;
  - $\{AOnline.DSI \mapsto AOnline.DSO\}$.

There are also other possible functional mappings as follows:

$$\{AOnline.ADI \mapsto AOnline.ADO;\\ AOnline.EVI \mapsto AOnline.EVO;\\ AOnline.AVI \mapsto AOnline.AVO;\\ AOnline.CRI \mapsto AOnline.CRO\};$$

which represent address verification, authorization and deposit with address verification, and so forth.

After comparing AID, CID (from specification S_C) to AOnline.ASI and AOnline.DSI, we can establish the following correspondences:

- AID vs. AOnline.ASI
  - Cardholder_name $\mapsto$ billing_phone
  - #Card $\mapsto$ cc_number
  - Exp_Date $\mapsto$ cc_exp_month, cc_exp_year
  - Bank_Acc $\mapsto$ merchant_email
  - Amount $\mapsto$ grand_total.
- CID vs. AOnline.DSI
  - Bank_Acc $\mapsto$ authorization and data from AOnline.ASI
  - Amount $\mapsto$ grand_total

We should note that values of the domains do not exactly match: billing_phone is used instead of Cardholder_name to identify cardholders; and merchant_email is used for Bank_Acc. Similarly, ok_Auth, ok_Capture, and BD_Update might correspond to the different values of TransactionStatus.

However, matching is possible since purpose is similar. Then, similarity is basically determined by analyzing semantics of concepts with respect to their use. This aspect introduces one of the key points that will be further discussed later. Just remember that some knowledge is needed to perform the matching, and consequently, calculate metrics.

Now, computing measures from Table 2 produces the following results;

$$CF_C(AcceptOnline) = 2,\ MF_C(AcceptOnline) = 0,\\ AF_C(AcceptOnline) = 4, \text{ and } CC_F(AcceptOnline) = 1.$$

These results indicate that the AcceptOnline component is 100% ($CC_F = 1$) functionally suitable, and thus a candidate for further evaluation during the filtering process. Measures also indicate that there are four added functions ($AF_C = 4$), which deserve more careful examination. However, after a closer look at those functions, we realized that many of them allow for possible variations of the authorization and capture functionalities – by considering "Address Verification" as complementary to other functions; for example, "Authorization with Address Verification".

On the other hand, "Credit" is actually adding functionality, since this function reimburses payments to the cardholder. This function has not been considered as relevant to our case because credit card payments cannot be cancelled. Taxpayers can call the credit card issuer or credit card payment service provider's customer service number to report problems such as unauthorized charges or concerns regarding payment errors.

Finally, the function "CK" checks the system – a useful supporting function but not domain-oriented, and therefore not relevant to our case.

Note here that different interpretations of what is relevant for a particular case, and what is considered as a supporting function introduces ambiguity to the calculation process. Calculations are affected by the level of detail (abstractness) of the required scenarios. Our second case illustrates this point more clearly. So, let us introduce our second example – the Energy Credit Card component.

4.2. The Energy Credit Card case

Energy Credit Card by Energy Programming Limited, is a COM object that allows users to accept credit card data via magnetic readers or keyed input. All data is validated to ensure authenticity and reduce chargebacks from the banking community. The Energy Credit Card component provides two functions described as follows:

(1) Functionality "Extract_Card_Data", which provides the ability to decode the magnetic data on the swipe card; and
(2) Functionality "Validate_Card_Details", which provides the ability to validate keyed entry data from other systems.

To accomplish both functionalities, input data is required as follows:

- Input: {surname, initials, salutation, card_number, card_type, startDate, expiryDate, issue}
- Output: {error_number, error_text}

As we easily can see, this second component does not provide the required functionality of our scenario. Although the component is classified as a member of the "Credit Card Authorization" group, functionalities show that only validation of credit card data is provided. Therefore, calculating measures from Table 2 produces the following results:

$CF_C(EnergyCreditCard) = 0$, $MF_C(EnergyCreditCard) = 2$, $AF_C(EnergyCreditCard) = 0$, and $CC_F(EnergyCreditCard) = 0$.

These results indicate that the Energy Credit Card component is 0% ($CC_F = 0$) functionally suitable, and we should not invest more time and effort in more evaluation. However, note that functionalities provided by the Energy Credit Card component might be part of the required functionality associated to the "Authorization" scenario. To make this point explicit, if necessary, evaluators should expose the different functionalities through a more detailed description of the required scenario; hence calculation of partially satisfied functionality would be possible. In our example, "Authorization" could be expressed as two sub-functions: "Credit Card Validation" and "Amount Authorization".

In this way, functionality supplied by the EnergyCreditCard would fit the functionality required by "Credit Card Validation"; thus calculating measures for the Energy Credit Card component would result in:

$CF_C(EnergyCreditCard) = 1$, $MF_C(EnergyCreditCard) = 2$, $AF_C(EnergyCreditCard) = 0$, and $CC_F(EnergyCreditCard) = 0.33$.

These results would indicate that the Energy Credit Card component might be a candidate to be combined along with other components to provide the required functionality (and not necessarily discharged).

Of course, decisions on how detailed a scenario should be depend on the requirements of a particular domain; i.e. components that do not provide the whole authorization procedure might not be useful in a particular case. We suppose here that balanced requirements among all stakeholders have been considered to provide the appropriated scenarios [24]. Note that domain knowledge to define scenarios is also an important influencing factor.

Now, let us consider a third component for our evaluation procedure.

4.3. The PaymentCardAssist case

The PaymentCardAssist component by Aldebaran, supports Email verification, event logging, data encryption, file compression, and payment card detail validation. The PaymentCard object within the DeveloperAssist Object Library validates payment card (credit, debit and charge card) information. The DeveloperAssist object library exposes a number of objects that can be used within ASP scripts. These objects are documented below.

- 7Email Validation. Provides the ability to check that an SMTP address is valid. It improves on typical JavaScript syntax checks by providing rigorous syntax checking and ensuring that the domain is configured to accept mail.
- Encryption. Performs data encryption and decryption, using the TwoFish algorithm from Counterpane Systems.
- Event Log. Writes to the Windows NT event log. Information, warning, error, success audit and failure audit events are all supported.
- File Compression. PKZip compatible file compression functions. The compressed data is fully compatible with other PKZip implementations.
- PaymentCard. Validates payment card (credit, debit and charge card) information.

The PaymentCard object does not provide authorization or clearing functionality, rather provides a means to validate payment information entered by a site visitor, before pursuing a full authorization.

After considering detailed data to be validated, our measures resulted as:

$CF_C(PaymentCard) = 0$, $MF_C(PaymentCard) = 2$, $AF_C(PaymentCard) = 4$, and $CC_F(PaymentCard) = 0$.

Or after considering a more detailed scenario, in which card data validation is made explicit, measures resulted as:

$CF_C(PaymentCard) = 1$, $MF_C(PaymentCard) = 2$, $AF_C(PaymentCard) = 4$, and $CC_F(PaymentCard) = 0.33$.

Here, we find a typical case in which added functionality is actually extra functionality. The metric $AF_C(PaymentCard)$ reveals four added functions, similarly to our AcceptOnline case. However, far from being complementary functions, this case illustrates the possibility of dealing with COTS components that are meant to be used in possibly different domains, and therefore, designed as "auxiliary" components, i.e. one in which we find different "oriented" functions, such as we see in this case – "validate" functions (email and payment card validation); "security" functions (encryption, event log); and "performance" functions (file compression). Thus, careful analysis of added functionality is necessary to make a decision on how to score it.

4.4. The CCValidate case

This component is not longer available; and unfortunately this is not the only case. The CCValidate COM object validated credit card numbers from applications and websites. CCValidate was an ActiveX DLL component that provided an Industry-approved validation algorithm (MOD 10 checksum). All major credit card types were supported, including VISA, Master Card, Amex, Discover, and more. Measures for this case resulted as:

$$CF_C(CCValidate) = 0,\ MF_C(CCValidate) = 2,$$
$$AF_C(CCValidate) = 0, \text{ and } CC_F(CCValidate) = 0.$$

This case illustrates another problem that composers usually face: the integration of vendor business continuity capability assessment as part of the filtering process. Although this analysis is not the main focus of our work, it deserves a few comments. The integration of business continuity metrics in the procurement process might be particularly interesting and should be related to the key elements of the filtering process. Incorporating the business continuity capability assessment at each phase of the selection process would help identify vulnerabilities, develop consequence management strategies, and implement mitigation strategies.

We would like to remark that the CCValidate component, and some others of our study, have dramatically changed during the last four months. In such a short period of time, some of them have migrated into a new version or even the provider company not longer exists. Further details on these changes will be described in Section 5.1.

4.5. The ComponentOne Studio Enterprise case

A special remark should be made on values assigned to the ComponentOne Studio Enterprise: this component is a combination of four individual components that support reporting, charting, data manipulation, and user interface capabilities for .NET, ASP.NET, and ActiveX applications. As we easily can see, this component essentially differs from the others in the group, and for this reason, additional functionality (AF_C) has not been scored. However, note that the ComponentOne Studio is classified as a "Credit Card Authorization" component by the ComponentSource organization.

Now, we will discuss the results particular to our case study in the following section.

5. Discussion

From 22 components surveyed in October 2004, we considered 12 for pre-filtering since the other 10 components only differed in terms of their implementations, preserving the same functionality.

Results of our calculations are shown in Table 5. Note that only four components provide the functionality required by our scenario. This fact would indicate that those components are pre-selected for more evaluation, since they are 100% functionally suitable.

Table 5
Measurement results for components of the "Credit Card Authorization" category

Component	CF_C	MF_C	AF_C	CC_F
AcceptOnline	2	0	4	**1**
EnergyCreditCard	0	2	0	0
PaymentCardAssist	0	2	4	0
CCProcessing	2	0	2	**1**
CCValidate	0	2	0	0
CreditCardPack	0	2	0	0
IBiz	2	0	2	**1**
InaCardCheck	0	2	0	0
IPWorks	2	0	0	**1**
LuhnCheck	0	2	0	0
SafeCard	0	2	0	0
ComponentOne Studio	0	2	—	0

Scenarios have been widely used during design as a method to compare design alternatives and to express the particular instances of each quality attribute important to the customer of a system. Scenarios differ widely in breadth and scope, and its appropriate selection is not straightforward. Our use of scenarios is a brief description of some anticipated or desired use of a system. The process of choosing scenarios for analysis forces designers to consider the future uses of, and changes to, the system. In some cases, this diversity of concerns produces fine-grained functionality described by scenarios, but coarse-grained functionality might be described as well.

As a consequence, our measures are affected by a particular scenario's description since calculation refers to the number of functions – without further discussion about their particular specification. For example, in our case study, "validation with address" and "reverse authorization" might be considered as part of an ordinary credit card authorization process. Assuming that, scores for added functionality (AF_C) would be decreased (only "credit" would be considered as added functionality). As another example, we could choose a more detailed description of the functionality and decompose Authorization into "Credit Card Validation" and "Amount Authorization". In this case, calculation of provided and missed functionality would be different and contribution (CC_F) would show which components partially contribute to reach credit card authorization.

Table 6 shows our measures considering the last two assumptions: (1) including "validation with/without address" and "reverse authorization" as part of the procedure, and (2) splitting Authorization into two processes – validation and authorization itself. By comparing scores from Tables 5 and 6 we illustrate the importance of standardizing the description of required functionality, as well as providing a more formal definition of scenarios.

Also, note that components providing all required functionality remain unchanged on both tables: only four components provide authorization and capture as required in our case (4/12 = 33%). It would indicate that searching a catalogue by category is not enough to find appropriate compo-

Table 6
Measurement results after changing scenarios

Component	CF_C	MF_C	AF_C	CC_F
AcceptOnline	3	0	1	1
EnergyCreditCard	1	2	0	0.33
PaymentCardAssist	1	2	4	0.33
CCProcessing	3	0	1	1
CCValidate	1	2	0	0.33
CreditCardPack	1	2	0	0.33
IBiz	3	0	1	1
InaCardCheck	1	2	0	0.33
IPWorks	3	0	0	1
LuhnCheck	1	2	0	0.33
SafeCard	1	2	0	0.33
ComponentOne Studio	0	3	—	0

nents. In our example, better categorizations would help distinguish credit card validation from authorization. Moreover, a better categorization would help avoid the situation where a component that does not provide any functionality (accordingly to the category), like ComponentOneStudio, is catalogued as a member of any of those classes.

Our measures indicate that four components are candidates to be accepted for more evaluation, i.e. the components are functionally suitable, but there is some additional functionality that could inject harmful side effects into the final composition. Identifying and quantifying added functionality are subject to similar considerations – essentially, the number of functions is a rough indicator that might be improved by weighting functionality; i.e. clearly the four functions added by PaymentCardAssist are different in scope and meaning from the functions added by the other components. However, just counting functions would help decide on which components the analysis should start.

Table 6 also shows that there are some candidates which are able to provide some required functionality – "credit card validation". But making this functionality more visible does not necessarily indicate the type of validation that actually is taking place, for example, whether or not a MOD10/Luhn check digit validation is carried out. Our measures are just indicators of candidates for further evaluation, on which additional effort might be invested. Nevertheless, our measures do not detect the best candidates at a first glance but a possible interesting set. The process guides calculations so ambiguity is decreased, but committed scenarios still depend on a particular system's requirements.

5.1. Lessons learned

(1) *Early measurement of functional suitability can reduce the number of candidates allowing a more objective value for decision making*. Our case study shows that measurement is possible at early stages by analyzing information of COTS components: from 12 COTS candidates, only four were pre-selected to be subjected to further analyses. However, we are aware that more experimental results are needed to be more conclusive about the applicability of our filtering process. Difficulties when applying the process might hinder its application, as the following lessons show.

(2) *Early detection of functionality requires that standards on how COTS components are documented be reinforced*. Information gathered from COTS candidates ranged from description of methods and properties in natural language to description by programming code. Certainly, this fact introduces ambiguity in some cases, complicates reading in some others, increases understanding effort, and makes the actual effectiveness of the filtering process dependent on the composer's expertise to detect candidates. Searching and gathering COTS component information must be supported by automatic tools. The lack of standard documentation not only hinders the matching of candidates, but also their classification and storing.

(3) *Requirements expressed as scenarios might facilitate searching and filtering, but a common understanding about the required level of detail (abstractness) must be specified*. Besides, classification of functionality to detect domain-oriented functions is necessary to separate concerns and identify added functionality, as the cases of AcceptOnline, EnergyCreditCard, CCProcessing, and PaymentCardAssist showed. The last case introduces some other considerations – not only functionality should be split to identify "auxiliary" functions, but also extra functionality should be weighted to indicate its relationship with the main function we are looking for (credit card payment in our case). Extra functionality, such as compressing files in PaymentCardAssist, requires careful examination. As a consequence, understanding functionality – and effectively identifying main/auxiliary/supporting functions – still depends on the scenario's specification and the composer's background and expertise. Similarly, careful examination is needed to identify partial functionality such as CeditCardPack, LuhnCheck, and other components supply. Here, further knowledge on how validation should be carried out would help candidate identification.

(4) *Composer's skills actually lead the search*. Of course, processes, techniques, and supporting tools are being defined to improve the filtering process, such as the ones defined in this paper. However, considering the previous lessons learned, they still rely on how a composer perceives requirements and offerings. Although every human-intensive process – such as the ones involved in software engineering – is always affected by human perception, ambiguity of the processes is usually decreased by using particular notations and standards. Unfortunately, they are not available for COTS component searching and filtering yet.

(5) *Assessing vendor's reliability is as much important as the identification of functional candidates itself*. We only have to look at our case study and the componentSource catalogue again[5]. From there and browsing the Web, we realized that the Bahs Software company is not a component supplier any more, and consequently AcceptOnline, CCProcessing,

[5] Accessed on February 25th, 2005. The following message is returned when accessing some of the components' former websites: "Sorry... The link that you have used is either incorrect or the product has been discontinued."

and CCValidate were withdrawn from the marketplace. Additionally, the Energy Programming Ltd. has changed its EnergyCreditCard component to the "Starfish EFT" component, which is an ActiveX component for validating credit card data; and PaymentCardAssist by Aldebaran has "evolved" into a complete suite called "Internet Commerce Toolkit". Then, from 11 COTS candidates in October 2004 (we exclude here ComponentOneStudio), only five of them remain available, and only one is suitable for our case – note that the IPWorks component is now only supplied as IBiz component. Then, our case study seems typical enough to sustain this lesson.

(6) *Classification is not straightforward.* As we have seen, our twelve components were catalogued as members of the "Credit Card Authorization" group by componentSource. From them, four components supply authorization and capture; seven components supply credit card validation; and one component – ComponentOneStudio – supplies many kinds of functionalities, not necessarily related to a credit card payment. Therefore, better classifications are needed. They might help facilitate the filtering process by using information about the catalogue itself. However, producing well-structured catalogues is not an easy task – component's granularity and several possibilities in classification make the process difficult.

5.2. Technical implications

As a first technical implication we argue that perception of what a COTS component is able to provide is not consistent. It means that the development of a COTS component market can be understood through the complexity of interchanging heterogeneous information between composers and suppliers. And here, "heterogeneous" may refer to different meanings for the same concept as well as different concepts addressing the same meaning. This conceptualization is currently researched as part of traditional ontology-based approaches. However, definitive solutions for solving semantic heterogeneity are far from achieved.

A second implication is on the need of homogeneous understanding to provide meaningful measurements – and therefore, to set a basis for improvement. In our research, this necessity was seen as an important pre-requisite for the calculations. Many research efforts are currently advocated to the development of ontologies and taxonomies to help identification and classification of COTS components [6,31–37]; and many others to better understand how a knowledge-based portal might be defined [36,38,39]. This paper notes the importance of both issues, which constitute the main technical implications.

6. Conclusion and future work

In the field of software engineering, there has been a strong emphasis on component-based software engineering, which means that software systems are developed based on pieces of software that can be independently deployed and developed. These pieces, or components, can either be developed by the company itself or more importantly, bought from an external component provider. This development in relation to software engineering methods has been accompanied by supplier companies entering the scene, offering commercial software components and suggesting that their products could provide customers with benefits such as cost reductions, higher quality and faster software development.

Then, software engineering practices have changed to face the challenge of identifying suitable COTS components from such a marketplace. As a consequence, during the last years, we witnessed the emergence and proliferation of COTS selection and integration methods. However, from the composer's perspective, although the possible benefits of the use of COTS selection methods are assumed to be many, in practice we face a lot of difficulties of varying degrees of importance and magnitude. Defining searching criteria is not such an easy task as it supposed to be; integration of candidates largely relies on exhaustive analysis of interface's compatibility; and few methods actually address measurement of candidates.

Our proposal was elaborated by defining particular techniques and metrics aiming at reducing the number of candidates by selecting some of them very quickly based on a brief review of key functional issues. Then, we have faced one of the current challenges during filtering: providing simple practices that help institutionalize formal selection procedures. Our functional suitability measures might drive a simple filtering process; however we are aware that lessons learned should not be neglected; then, we should base our filtering procedure on a more formal description of COTS components' properties.

To do so, we have firstly classified some current efforts that try to describe COTS components [40]. In this context, some proposals use description logics to develop an ontology for matching requested and provided components, or propose taxonomies for classification [31,33,34,36,37,41,42]; others suggest extending the identification stage with a learning phase, which provides support to the COTS component discovery process [6,43]; and some other approaches try to measure the semantic distance between required and offered functionality [23,44], although these measures usually need detailed information as input to the calculations.

Secondly, we have looked at some current catalogs on the Web. However, in spite of several catalogues' existence, a common description model is still an open issue – currently some portals only get an overview of the technical and legal features of the software; others focus only on a particular type of component; and others only contain Web services descriptions. Motivated by this situation, the eCots[6] project aims at developing an open information portal for COTS components, in which we deal with information about products, and possibly between their users, or between users and producers [39].

[6] www.ecots.org

Particularly, the information of this portal will give us the opportunity to use (and extend) our research as a contribution to guide the filtering process. Hopefully, empirical cases will be conducted in the near future based on the use of the eCots platform. However, several aspects need further discussion before using such a portal. For example, what kind of information will characterize a COTS product? Should this information be based on properties of COTS products? Are there many COTS products sharing similar properties? Can they be generalized?

Then, from limitations discussed above, lessons learned, and main implications, we aim at improving the measurement phase by including ontology- and taxonomy-based matching. In this way, a more formal definition and standard information of components' services would facilitate filtering.

Acknowledgements

We thank anonymous reviewers for their valuable comments. This work is partially supported by the CyTED (Ciencia y Tecnología para el Desarrollo) project 506PI287 CompetiSoft; the UNComa project 04E/059, and the MAS project supported by the Dirección General de Investigación of the Ministerio de Ciencia y Tecnología, Spain (TIC 2003-02737-C02-02).

References

[1] A. Cechich, M. Piattini, A. Vallecillo (Eds.), Component-Based Software Quality: Methods and Techniques, Lecture Notes in Computer Science, vol. 2693, Springer-Verlag, Berlin, 2003.

[2] J. Li, F. Bjornson, R. Conradi, V. Kampenes, An empirical study on COTS component selection process in Norwegian IT companies, in: Proceedings of the First ICSE International Workshop on Models and Processes for the Evaluation of COTS Components, IEE, Edinburgh, Scotland, 2004, pp. 27–30.

[3] M. Torchiano, M. Morisio, Overlooked aspects of COTS-based development, IEEE Software 21 (2) (2004) 88–93.

[4] M. Ochs, D. Pfahl, G. Chrobok-Diening, Nothhelfer-Kolb, A method for efficient measurement-based COTS assessment and selection – method description and evaluation results, Tech. Rep. IESE-055.00/E, Fraunhofer Institut Experimentelles Software Engineering, 2000.

[5] C. Abts, COTS-based systems (CBS) functional density – a heuristic for better CBS design, in: Proceedings of the 1st International Conference on COTS-Based Software Systems, Lecture Notes in Computer Science, vol. 2255, Springer-Verlag, Orlando, Florida, 2002, pp. 1–9.

[6] A. Bianchi, D. Caivano, R. Conradi, L. Jaccheri, M. Torchiano, G. Visaggio, COTS products characterization: proposal and empirical assessment, in: Proceedings of ESERNET 2001–2003Lecture Notes in Computer Science, vol. 2765, Springer-Verlag, Orlando, Florida, 2003, pp. 233–255.

[7] L. Holmes, Evaluating COTS Using Function Fit Analysis, Q/P Management Group, INC – <http://www.qpmg.com/>.

[8] A. Cechich, M. Piattini, On the measurement of COTS functional suitability, in: Proceedings of the Third International Conference on COTS-Based Software Systems, Lecture Notes in Computer Science, vol. 2959, Springer-Verlag, Los Angeles, USA, 2004, pp. 31–40.

[9] M. Bertoa, J. Troya, A. Vallecillo, 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, 2003, pp. 25–30.

[10] P. Ulkuniemi, V. Seppäanen, COTS component acquisition in emerging markets, IEEE Software 21 (6) (2004) 76–82.

[11] A. Cechich, M. Piattini, Managing COTS components using a Six Sigmabased process, in: Proceedings of the Fifth International Conference on Product Focused Software Process Improvement, Lecture Notes in Computer Science, vol. 3009, Springer-Verlag, Nara, Japan, 2004, pp. 556–567.

[12] A. Cechich, M. Piattini, Filtering COTS components through an improvement-based process, in: Proceedings of the Fourth International Conference on COTS-Based Software Systems, Lecture Notes in Computer Science, vol. 3412, 2005, pp. 112–121.

[13] H. Lipson, N. Mead, A. Moore, Can we ever build survivable systems from COTS components? in: Proceedings of CAiSE 2002LNCS, vol. 2348, Springer-Verlag, Orlando, Florida, 2002, pp. 216–229.

[14] B. Boehm, A spiral model of software development and enhancement, IEEE Computer (1998) 61–72.

[15] C. Albert, L. Brownsword, Evolutionary process for integrating COTS-based systems (EPIC): an overview, Tech. Rep. 20030TR-009, SEI, 2002.

[16] R. Biehl, Six Sigma for software, IEEE Software 8 (2) (2004) 68–70.

[17] J.D. Feo, Z. Bar-El, Creating Strategic change more efficiently with a new Design for Six Sigma process, Journal of Change Management 3 (1) (2002) 60–80.

[18] A. Gack, K. Robinson, Integrating improvement initiatives: connecting Six Sigma for software, CMMI, personal software process and team software process, Software Quality Journal 5 (4) (2003) 5–13.

[19] C. Tayntor, Six Sigma Software Development, AUERBACH Publications, 2003.

[20] I. Jacobson, G. Booch, J. Rumbaugh, The Unified Development Process, Addison-Wesley, Boston, MA, 1999.

[21] C. Ncube, N. Maiden, Guiding parallel requirements acquisition and COTS software selection, in: Proceedings of the IEEE International Symposium on Requirements Engineering, 1999, pp. 133–141.

[22] N. Maiden, C. Ncube, Acquiring COTS software selection requirements, IEEE Software 15 (2) (1998) 46–56.

[23] R. Alexander, M. Blackburn, Component assessment using specification-based analysis and testing, Tech. Rep. SPC-98095-CMC, Software Productivity Consortium, 1999.

[24] A. Cechich, M. Piattini, Balancing Stakeholders preferences on measuring COTS component functional suitability, in: Proceedings of the Sixth International Conference on Enterprise Information Systems, Porto, Portugal, 2004, pp. 115–122.

[25] ISO/IEC, Software engineering – COSMIC-FFP – a functional size measurement method, International Standard ISO/IEC 19761, International Standards Organization, Geneva, Switzerland, 2003.

[26] A. Cechich, M. Piattini, Quantifying COTS component functional adaptation, in: Proceedings of the Eight International Conference on Software ReuseLecture Notes in Computer Science, vol. 3107, Springer-Verlag, Madrid, Spain, 2004, pp. 195–204.

[27] R. Stake, The Art of Case Study Research, Sage Publications, 1995.

[28] K. Eisenhardt, Building theories from case study research, Academy of Management Review 14 (4) (1989) 532–550.

[29] C. Rolland, C. Souveyet, C.B. Achour, Guiding goal modelling using scenarios, IEEE Transactions on Software Engineering 24 (12) (1998) 1055–1071.

[30] A. Cechich, M. Piattini, Early detection of COTS functional suitability for an E-payment case study, in: 7th International Conference on Enterprise Information Systems, Miami, USA, 2005.

[31] T. Asikainen, T. Soinien, T. Mäannistö, A Koala-based ontology for configurable software product families, in: Proceedings of the Workshop on Configuration in Conjunction with the Eighteenth International Joint Conference on Artificial Intelligence, 2003.

[32] C. Ayala, P. Botella, X. Franch, On goal-oriented COTS taxonomies construction, in: Proceedings of the Fourth International Conference on COTSBased Software Systems, Lecture Notes in Computer Science, vol. 3412, 2005, pp. 90–100.

[33] R. Braga, M. Mattoso, C. Werner, The use of mediation and ontology technologies for software component information retrieval, in: Proceedings of the 2001 Symposium on Software Reusability: Putting Software Reuse in Context, ACM Press, 2001, pp. 19–28.
[34] J. Carvallo, X. Franch, C. Quer, M. Torchiano, Characterization of a taxonomy for business applications and the relationships among them, in: Proceedings of the Third International Conference on COTS-Based Software SystemsLecture Notes in Computer Science, vol. 2959, Springer-Verlag, Orlando, Florida, 2004, pp. 221–231.
[35] P. Fettke, P. Loos, Specification of business components, in: Proceedings of NetObjectDays 2002Lecture Notes in Computer Science, vol. 2591, Springer-Verlag, 2003, pp. 62–75.
[36] S. Overhage, UnSCom: a standardized framework for the specification of software components, in: 5th Annual International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World (NODe 2004)Lecture Notes in Computer Science, vol. 3263, Springer-Verlag, 2004, pp. 169–184.
[37] C. Pahl, Ontology-based description and reasoning for component-based development on the Web, in: Proceedings of SAVCBS'03-ESEC/FSE'03 Workshop, ACM, 2003.
[38] M. Hristozova, L. Sterling, Experiences with ontology development for value- added publishing, in: OAS 2003 – Workshop on Ontologies in Agent Systems, 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems, 2003. URL <http://oas.otago.ac.nz/oas2003//>.
[39] J.-C. Mielnilk, B. Lang, S. Lauriére, J.-G. Schlosser, V. Bouthors, eCots platform: an inter-industrial initiative for COTS-related information sharing, in: Proceedings of the Second International Conference on COTS-Based Software Systems, Lecture Notes in Computer Science, vol. 2580, Springer-Verlag, 2003, pp. 157–167.
[40] A. Cechich, A. Requilé, J. Aguirre, J. Luzuriaga, Trends on COTS component identification, in: Proceedings of the 5th IEEE International Conference on COTS-Based Software Systems, IEEE Computer Science Press, 2006, pp. 90–99.
[41] T. Asikainen, T. Soinien, T. Männistö, Representing software product family architectures using a configuration ontology, in: Proceedings of the Workshop on Configuration in conjunction with the 15th European Conference on Artificial Intelligence (ECAI-2002), 2002.
[42] C. Pahl, An ontology for software component matching, in: Proceedings of the Sixth International Conference on Fundamental Approaches to Software EngineeringLecture Notes in Computer Science, vol. 2621, Springer-Verlag, 2003, pp. 6–21.
[43] L. Jaccheri, M. Torchiano, A Software Process Model to Support Learning of COTS Products, Tech. rep., IDI NTNU, 2002.
[44] L. Jilani, J. Desharnais, Defining and applying measures of distance between specifications, IEEE Transactions on Software Engineering 27 (8) (2001) 673–703.