EARLY DETECTION OF COTS FUNCTIONAL SUITABILITY FOR AN E-PAYMENT CASE STUDY

Alejandra Cechich
Departamento de Ciencias de la Computación, Universidad del Comahue, Neuquén, Argentina
Email: acechich@uncoma.edu.ar

Mario Piattini
Escuela Superior de Informática, Universidad de Castilla-La Mancha, Ciudad Real, España
Email: Mario.Piattini@uclm.es

Keywords: Component-Based System Assessment, COTS components, Software Quality.

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 currently carried out dealing with unstructured information on the Web, which makes the evaluation process highly costing when applying complex evaluation criteria. To facilitate the process, in this paper we introduce an early measurement procedure for functional suitability of COTS candidates, and we illustrate the proposal by evaluating components for an e-payment case study.

1 INTRODUCTION

COTS-Based System Development is now recognised as the way forward in building software architectures that can operate in advanced distributed, intranet, and Internet environments. In essence, using components to build systems reduces complexity because composers do not need to know how a component works internally. They only need to know what the component is and the services it provides. Ideally, most of the application developer's time is spent integrating components. Components become unwieldy when combined and re-combined in large-scale commercial applications. What are needed are ensembles of components that provide major chunks of application functionality that can be snapped together to create complete applications.

COTS component filtering is to decide which components should be selected for more detailed evaluation. Decisions are driven by a variety of factors – foremost are several design constraints that help define the range of components. So a balance is struck, depending upon the level of abstraction, complexity of the component, goals and criteria, and so forth. Some methods include qualifying thresholds for filtering. For example, during the activity "Collect Measures" of the COTS Acquisition Process (Ochs et al., 2000), data according to a measurement plan are collected on a set of COTS software alternatives. Data are used in the filtering activity to eliminate those COTS alternatives that are unacceptable for use.

Identification of COTS candidates is a complex activity itself. It implies not only dealing with an impressive number of possible candidates but also with unstructured information that requires a careful analysis. In this context, some proposals use description logics to develop an ontology for matching requested and provided components (Braga et al., 1999; Pahl, 2003); others suggest extending the identification stage with a learning phase, which provides support to the COTS component discovery process (Jaccheri and Torchiano, 2002). Some other approaches try to measure the semantic distance between required and offered functionality (Alexander and Blackburn, 1999; Jilani and Desharnais, 2001) but these measures usually need detailed information as input to the calculations.

In addition to learning and classification issues, a filtering process is concerned with the pre-selection of candidates. It actually takes place by matching several properties of COTS components, including some inexact matching. Moreover, there are some cases where goals cannot be entirely satisfied without considerable product adaptation and other

cases where these goals must be resigned to match product features (Alves and Filnkestain, 2002); (Cooper and Chung, 2002).

As a possible improvement, the Six Sigma approach has been suggested selecting packaged software (Tayntor, 2002); however the evaluation mainly relies on the information provided by demos and additional documentation of the software. Then, the lack of measures makes this process perfectible.

Along these lines, our approach based on Six-Sigma precepts, focuses on fact-based decisions, teamwork, and measurement as a way of driving the identification and filtering process (Cechich and Piattini, 2004a; Cechich and Piattini 2004b).

We refer to a component-based system as a system that uses at least one component in conjunction with other components, legacy systems, and other pieces of software – including COTS components – to satisfy user's requirements. This concept is introduced to emphasize the fact that the output from the system satisfies the user's requirements by using the functionality supplied by at least one COTS component. Particularly, we consider functional suitability as the main aspect to be measured; however, measures should be expressed in such a way that calculation is possible at early stages.

Our proposal aims at improving the filtering process by performing three steps: (1) a "commitment" step, which produces a committed required specification of a component; (2) a "pre-filtering" step, in which COTS candidates are pre-selected according to their functional suitability; and (3) a "filtering" step, in which architectural semantics adaptability produces an indicator of stability that serves as a basis for the final candidate filtering. In this paper, we particularly address the second step ("pre-filtering"), in which functional suitability measures are calculated and analysed.

Metrics for COTS based systems are emerging from the academic and industrial field (Martín-Albo et al., 2003). However, many of these definitions do not provide any guideline or context of use, which makes metric's usability dependable on subjective applications. Measures are not isolated calculations with different meanings; on the contrary, capability of measures is strongly related to the process of calculating and providing indicators based on the measures. Our approach intends to define a filtering process in which measures are included as a way of providing more specific values for comparison. At the same time, the process guides the calculation, so ambiguity is decreased.

Among other relationships, resulting measures are related to the artefact to be measured. In our approach, the artefact is expressed as functionality required by a particular application, and functionality offered by COTS candidates. Generally speaking, both cases are subject to analysing information that is modelled and weighted by people – composers or integrators on one side, and component's suppliers on the other. Different interpretations, perceptions, and judgements are then affected by the expressiveness of information. Nevertheless, our comparisons are abstract-level definitions, which allow us to customize the filtering process by instantiating the calculation procedure according to different contexts of use.

Since information needed to compute the measures depends on how COTS suppliers document COTS component's functionality (Bertoa et al., 2003), and how requirements are specified, in this paper we illustrate how metrics might be calculated by measuring functional suitability on COTS candidates for an E-payment case study.

In section 2 we briefly introduce our compact suite of measures (Cechich and Piattini, 2004c) that should be used during the pre-filtering process. Then, section 3 shows how measures might be applied to our case and provides some discussion. A final section addresses conclusions and topics for further research.

2 MEASURING FUNCTIONAL SUITABILITY

In the previous section, we have emphasized the fact that a system should satisfy the user's requirements by using the functionality supplied by at least one COTS component. Then, given a specification S_C for an abstract component type C, a candidate component K to be a concrete instance of C must conform to the interface and behaviour specified by S_C. Mappings in S_C, which represent the different required functionalities, are established between input and output domains. We focus on incompatibilities derived from functional differences between the specification in terms of mappings of a component K_i (S_{Ki}) and the specification in terms of mappings of S_C.

Our measures have been defined to detect domain compatibility as well as functional suitability. Let us briefly clarify this point: domain compatibility measures show that there are some candidate components able to provide some functionality. However, we cannot be certain of the amount of functionality that is actually provided – matching input data does not certify that output data match too. Therefore, even a component might be full domain compatible, there is still another set of measures to be applied in order to determine the functional suitability.

Let us illustrate the measurement procedure by using an credit card payment system as an example. We suppose the existence of some scenarios describing the two main stages of the system – *authorization* and *capture*. 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. Scenarios will provide an abstract specification of the mappings of S_C that might be composed of:

- Input domain:
(AID) Auth_IData{#Card, Cardholder_Name, Exp-Date};
(CID) Capture_Idata{Bank_Account, Amount}.
- Output domain:
(AOD) Auth_Odata{ok-Auth};
(COD)
Capture_Odata{ok_Capture, DB_Update}.
- Mapping: {AID → AOD};{CID → COD}

Suppose we pre-select two components to be evaluated, namely K_1 and K_2 respectively. A typical situation for inconsistency in the functional mappings between S_{K1}, S_{K2} and S_C is illustrated in Figure 1, where dashed lines indicate (required) mappings with respect to S_C, and the solid lines are (offered) mappings with respect to S_{K1} (grey) and S_{K2} (black). Note that the input domain of the component K_1 does not include all the values that the specification S_C requires, i.e. the capture functionality is not provided. Besides, the input domain of the component K_2 includes more values than the required by S_C, although the mapping satisfies the required functionality. We should also note that there is another functionality provided by K_2, i.e. {Taxes → Statistics}, which might inject harmful effects to the final composition.

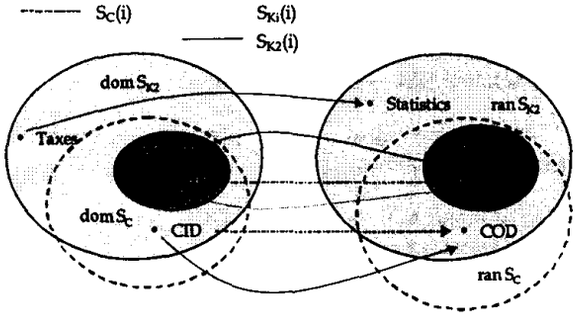

Figure 1: Functional mappings of S_C and S_{K1}/S_{K2}

Table 1: Description of 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 *S*
MF_C Missed Functionality	The number of functional mappings required by S_C and NOT provided by S_K in the scenario *S*.
AF_C Added Functionality	The number of functional mappings NOT required by S_C and provided by S_K in the scenario *S*.
CC_F Component Contribution	Percentage in which a component contributes to get the functionality required by S_C in the scenario *S*.
Solution-Level	
SN_{CF} Candidate Solution	The number of components that contribute with compatible functionality to get the requirements of S_C in the scenario *S*.
CF_S Compatible Functionality	The number of functional mappings provided by *SN* and required by S_C in the scenario *S*.
MF_S Missed Functionality	The number of functional mappings required by S_C in the scenario *S* and NOT provided by *SN*.
AF_S Added Functionality	The number of functional mappings NOT required by S_C in the scenario *S* and provided by *SN*.
SC_F Solution Contribution	Percentage in which a solution contributes to get the functionality required by S_C in the scenario *S*.

Our measures on functional suitability have been classified into two different groups: component-level measures and solution-level measures. The first group of measures aims at detecting incompatibilities on a particular component *K*, which is a candidate to be analysed. However, it could be the case that we need to incorporate more than one component to satisfy the functionality required by the abstract specification S_C. In this case, the second group of measures evaluates the functional suitability of all components that constitute the candidate solution.

Table 1 lists our suite of functional suitability measures. We refer the reader to (Cechich and Piattini, 2004c) for their formal definition. Solution-level metrics are listed here for completeness reasons, since our case study only needs to apply component-level measures; i.e. combination of

components from the marketplace is not necessary to get the required functionality, therefore a solution-level analysis is not required.

3 MEASURING COTS CANDIDATES: A CASE STUDY

Scenarios describing the two main stages of a credit card payment, as we introduced in the previous section, represent here a credit card (CCard) payment system, which provide an abstract specification of the input (AID, CID) and output domains (AOD, COD) of a component C, and their corresponding mappings.

After a quick browse on the Web as a COTS repository, we chose COTS components catalogued by the ComponentSource organization (www.componentsource.org) as members of the "Credit Card Authorization" group. Following, we introduce some examples of our analysis.

Firstly, we chose one component – AcceptOnline by Bahs Software – as a candidate to provide the required functionality. 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.

Table 2: Required Fields by Transaction Type

Field	CK	AD	AS	ES	EV	AV	DS	CR
authorization							Y	
billing_address1; billing address2		Y			Y	Y		
billing_zip		Y			Y	Y		
billing_pone		Y	Y	Y	Y	Y		
cc_number; cc_exp_month; 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_numer							Y	Y
....								...

Methods of AcceptOnline are specified in terms of their main focus and required input. Particularly, the SendPacket method is used to send the transaction info to the ECHOOnline server, and required properties should be filled as shown in Table 2 (requirements for CR are partially listed).

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

– Input domain:
(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}.

– Output domain:
(AOnline.ASO) {TransactionStatus};
(AOnline-DSO) {TransactionStatus}.

– Mapping:
{AOnline.ASI → AOnline.ASO;
AOnline.DSI → AOnline.DSO} .

There are also other possible functional mappings as follows:

{AOnline.ADI → AOnline.ADO;
AOnline.EVI → AOnline.EVO;
AOnline.AVI → AOnline.AVO;
AOnline.CRI → AOnline.CRO},

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

For brevity reasons, we assume here that input domain compatibility measures have indicated that the AcceptOnline component is a candidate for further evaluation – after comparing AID, CID (from specification S_C) to AOnline.ASI and AOnline.DSI. We should note that values of the input domain do not exactly match: billing_phone is used instead of cardholder_name to identify cardholders; and merchant_email is used for Bank_id. Similarly, ok_Auth, ok_Capture, and BD_Update might correspond to the different values of TransactionStatus. However, in all cases matching is possible since purpose is similar. Then, similarity is basically determined by analysing semantics of concepts with respect to their use.

Now, computing measures from Table 1 produces the following results:

$CF_C = 2$; $MF_C = 0$; $AF_C = 4$; and $CC_F = 1$

These results indicate that the AcceptOnline component has proved being 100% ($CC_F = 1$) functionally suitable, and thus a candidate for further evaluation during the filtering process – for example by analysing size and complexity of adaptation. Measures also indicate that there are four added functions ($AF_C = 4$), which deserve more careful examination.

Let us analyse a second component from the same group, i.e. catalogued as a member of "Credit Card Authorization". This time, we have chosen the Energy Credit Card component by Energy Programming as the candidate to provide the required functionality.

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 1 would produce the following results:

$CF_C = 0$; $MF_C = 2$; $AF_C = 0$; and $CC_F = 0$

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 "Credit Card Validation" and "Amount Authorization". In this way, calculating measures for the Energy Credit Card component would result in:

$CF_C = 1$; $MF_C = 2$; $AF_C = 0$; and $CC_F = 0.33$

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 an scenario should be depend on requirements on 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 (Cechich and Piattini, 2004b).

Now, let us consider a third component for our evaluation procedure: the PaymentCardAssist component by Aldebaran, that supports e-mail 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 PaymentCard object does not provide authorization or clearing functionality, but 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, we can see that our measures will result as:

$CF_C = 0$; $MF_C = 2$; $AF_C = 4$; and $CC_F = 0$;

or after considering a more detailed scenario, in which card data validation is made explicit, measures will result as:

$CF_C = 1$; $MF_C = 2$; $AF_C = 4$; and $CC_F = 0.33$

Finally, let us consider another component from the same group – the CCProcessing component by Bahs Software. It supports the authorization, settlement (capture) , and credit/refund operations.

"Authorization" is divided into "PerformAuthorization" and "AddToBatch" operations, meanwhile "Capture" corresponds to the "PerformSettlement" operation. Transaction descriptions are presented as follows:

- "PURCHASE": Standard purchase transaction (In "card not present" mode);
- "PURCHASE_TRACK1": Purchase transaction in "card present" mode. Track1 property should be set for such transaction type.
- "VALIDATE_CARD": Card authentication to determine only if a card has been reported lost or stolen.
- "REVERSE_AUTHORIZATION": On-line Authorization Reversal.
- "REVERSE_SETTLEMENT": Store & Forward Authorization Reversal.
- "CREDIT": Credit/refund operation.

By analysing input and output domains of CCProcessing, we have identified mappings that cover the functionalities described by our scenario. Considering "credit" and address validation (part of "validate card") as additional functionality (reverse authorization and reverse settlement might be considered as part of a "Cancel" operation), measurement results might be expressed as:

$CF_C = 2$; $MF_C = 0$; $AF_C = 2$; and $CC_F = 1$

A similar treatment was applied to evaluate the other components in the group. From 22 components, we consider 12 for analysis since the other 10 components differ only in terms of their implementations, preserving the same functionality.

Results of our calculations are shown in Table 3. 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. 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 readers easily can see, this component essentially differs from the others in the group; however it is classified as a "Credit Card Authorization" component. For this reason, additional functionality (AF_C) has not been scored.

Table 3: Measurement results for components in the "Credit Card Authorization" category

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

3.1 Discussion

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. We emphasize the use of scenarios appropriated to all roles involving a system. The evaluator role is one widely considered but we also have roles for the system composer, the reuse architect, and others, depending on the domain.

The process of choosing scenarios for analysis forces designers to consider the future uses of, and changes to, the system. It also forces to consider non-functional properties that should be properly measured during the COTS selection process. 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 CCard system, "validation with address" and "reverse authorization" could 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 "Credit Card 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 4 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 Table 3 and Table 4 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 appropriated components. In our example, better categorizations would help distinguish credit card validation from authorization. Moreover, a better categorization would help avoid that a component that does not provide any functionality (accordingly to the category), like ComponentOneStudio, be catalogued as a member of any of those classes.

Table 4: Measurement results after changing scenarios

Component	CF_C	MF_C	AF_C	CC_F
AcceptOnline	3	0	1	1
CCProcessing	3	0	1	1
CCValidate	1	2	0	0.33
CreditCardPack	1	2	0	0.33
EnergyCreditCard	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
PaymentCardAssist	1	2	4	0.33
SafeCard	1	2	0	0.33
ComponentOneStudio	0	3	***	0

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 – the number of functions essentially is a rough indicator that might be improved by weighting functionality; i.e. clearly the four functions added by the component PaymentCardAssist are different in scope and meaning from the other added functions. However, just counting functions would help decide on which components the analysis should start.

Table 4 also shows that there are some candidates which are able to provide some required functionality – "credit card validation". But making this functionality more visible 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. A process guides calculations so ambiguity is decreased (Cechich and Piattini, 2004a), but committed scenarios still depend on particular system's requirements.

Besides, there are another types of analysis the component should be exposed before being eligible as a solution – such as analysis of non-functional properties, analysis of vendor viability, and so forth (Ballurio et al., 2002). Our set of measures are only providing a way of identifying suitable components from a functional point of view. We might provide a more precise indicator when calculating the maintenance equilibrium value as introduced in (Abts, 2002): "*Maximise the amount of functionality in your system provided by COTS components but using as few COTS components as possible*".

A final remark brings our attention into the necessity of balancing required and offered functionality during COTS-based developments. After analysing candidates, we might also change our expectations on finding appropriated components. In this case, we could potentially resign most of our expectations on a particular requirement letting offered services prevail. For example, we could keep some of the alternative services resigning others whether COTS candidates are hard to find or adapt. An additional measure on modifiability of goals (Cechich and Piattini, 2004b) would help detect the degree in which certain functionality can be changed when selecting COTS components. Of course, we could also decide not to select components at all, and build a solution from scratch.

4 CONCLUSION

We have briefly presented some measures for determining functional suitability of COTS candidates by applying the calculations on a case study. It showed how COTS information may be mapped onto our measurement model leading to an early value for decision making.

However, differences in COTS component documentation make evaluation harder. Our application clearly remarks the importance of standardising COTS component documentation and analysing the diverse ways of structuring COTS component's information to facilitate functional matching detection. However, successful matching also depends on how functional requirements are specified. Then, a formal procedure for identification of candidates should be defined to make the process cost-effectively.

Constraints on the component's use and constraints relative to a context might be also useful to be considered. These aspects would indicate that providing more complex classifications, such as taxonomies of components, would help catalogue them in a marketplace. Additionally, more complex descriptions might be provided by using ontologies and contexts. Along these lines, our future work aims at defining some guidelines and hints on the searching and learning process of COTS component candidates.

ACKNOWLEDGMENTS

This work was partially supported by the CyTED project VII-J-RITOS2, by the UNComa project 04/E059, and by the MAS project supported by the Dirección General de Investigación of the Ministerio de Ciencia y Tecnología (TIC 2003-02737-C02-02).

REFERENCES

Abts C. COTS-Based Systems (CBS) Functional density - A Heuristic for Better CBS Design, 2002. In *Proceedings of the First International Conference on COTS-Based Software Systems*, Springer Verlag LNCS 2255, pages 1–9.

Alexander R. and Blackburn M., 1999. Component Assessment Using Specification-Based Analysis and Testing. *Technical Report SPC-98095-CMC, Software Productivity Consortium.*

Alves C. and Filkenstein A., 2002. Challenges in COTS-Decision Making: A Goal-Driven Requirements Engineering Perspective. In *Proceedings of the Fourteenth International Conference on Software Engineering and Knowledge Engineering*, SEKE'02.

Ballurio K., Scalzo B., and Rose L, 2002. Risk Reduction in COTS Software Selection with BASIS. In *Proceedings of the First International Conference on COTS-Based Software Systems*, ICCBSS 2002, Springer-Verlag LNCS 2255 , pp. 31-43.

Bertoa M., Troya J., and Vallecillo A., 2003. A Survey on the Quality Information Provided by Software Component Vendors. In *Proceedings of the ECOOP QAOOSE Workshop*

Braga R., Mattoso M., and Werner C., 2001. The use of mediation and ontology for software component information retrieval. In *Proceedings of the 2001 Symposium on Software Reusability: putting software reuse in context,* ACM Press, pp. 19-28.

Cechich A. and Piattini M., 2004a. Managing COTS Components using a Six Sigma-Based Process. In *Proceedings of the 5th International Conference on Product Focused Software Process Improvement*, PROFES 2004, volume 2009 of LNCS, Springer-Verlag, pp.556-567.

Cechich A. and Piattini M., 2004b. Balancing Stakeholder's Preferences on Measuring COTS Component Functional Suitability. In *Proceedings of the 6th International Conference on Enterprise Information Systems*, ICEIS 2004, pp. 115-122.

Cechich A. and Piattini M., 2004c. On the Measurement of COTS Functional Suitability. In *Proceedings of the 3rd International Conference on COTS-based Software Systems*, ICCBSS 2004, volume 2959 of LNCS, Springer-Verlag, pp. 31-40.

Cooper K. and Chung L., 2002. A COTS-Aware Requirements Engineering and Architecting Approach: Defining System Level Agents, Goals, Requirements and Architecture, *Technical Report UTDCS-20-02, Department of Computer Science, The University of Texas at Dallas.*

Jaccheri L. and Torchiano M., 2002. A Software Process Model to Support Learning of COTS Products. *Technical Report, IDI NTNU.*

Jilani L. and Desharnais J., 2001. Defining and Applying Measures of Distance Between Specifications. *IEEE Transactions on Software Engineering*, 27(8):673—703 .

Martín-Albo J., Bertoa M., Calero C., Vallecillo A., Cechich A., and Piattini M., 2003. CQM: A Software Component Metric Classification Model. In *Proc. of the 7th ECOOP Workshop QAOOSE 2003*, pages 54-60, Darmstadt, Germany.

Ochs, D. Pfahl, G. Chrobok-Diening, and Nothhelfer-Kolb, 2000. A Method for Efficient Measurement-based COTS Assessment and Selection - Method Description and Evaluation Results. *Technical Report IESE-055.00/ E, Fraunhofer Institut Experimentelles Software Engineering.*

Pahl C., 2003. An Ontology for Software Component Matching. *In Proceedings of the Sixth International Conference on Fundamental Approaches to Software Engineering*, volume 2621 of LNCS, Springer-Verlag, pp. 6-21.

Tayntor C., 2002. *Six Sigma Software Development.* Auerbach Publications.