# On the Measurement of COTS Functional Suitability

Alejandra Cechich[1] and Mario Piattini[2]

[1] Departamento de Ciencias de la Computación
Universidad Nacional del Comahue, Buenos Aires 1400
Neuquén, Argentina
acechich@uncoma.edu.ar
[2] Grupo Alarcos, Escuela Superior de Informática
Universidad de Castilla-La Mancha, Paseo de la Universidad 4
Ciudad Real, España
Mario.Piattini@uclm.es

**Abstract.** Within the last years both researchers and practitioners alike have moved beyond establishing COTS quality as an important field to resolving CBS quality problems. However, the science of CBS quality has not yet advanced to the point where there are standard measurement methods, and few enterprises routinely measure COTS quality. Here, a suite of measures is presented to address this problem within a COTS-based software measurement activity. Our measures are based on a formally defined component-based model, aiming at expressing and measuring some aspects of component integrations. Measures are in terms of provided and required services, hence functional suitability might be quantified.

## 1 Introduction

Software project managers need to make a series of decisions at the beginning of and during projects. Because software development is such a complex and diverse process, predictive models should guide decision making for future projects. This requires having a metrics program in place, collecting project data with a well-defined goal in a metrics repository, and then analysing and processing data to generate models. According to the proposal in [11], metrics can guide risk and quality management, helping reduce risks encountered during planning and execution of CBSD.

Metrics let developers identify and quantify quality attributes in such a way that risks encountered during COTS selection are reduced. For example, the QESTA approach to evaluate COTS components [8] defines for each desired quality one or more metrics, either symbols or numbers. Then, the selected candidate components are each measured against the metrics previously identified. As another example, based on the ISO/IEC 9126 Standard for software product evaluation [10], the proposal in [5] restricts the set of features applicable on COTS components and defines two classes of measurable features: run-time measured features and life cycle measured features.

R. Kazman and D. Port (Eds.): ICCBSS 2004, LNCS 2959, pp. 31–40, 2004.

All CBS projects require a cost estimate before actual developments can proceed. Usually, the qualities of the desired COTS components are not directly measurable but are instead vague statements about like “acceptable performance”, “small size”, and “high reliability”. Thus, most cost estimates for CBS developments are based on rules of thumb involving some size measure, like adapted lines of code, number of function points added/updated, or more recently, functional density [1, 7, 9]. In practical terms, rules such as functional density imply that there must be a way of comparing one CBS design to another in terms of their functionality, there must be possible to split functionality delivered via COTS from that delivered from scratch, and there must be a way to clearly identify different COTS functionalities [1].

On the other hand, the model introduced in [2] 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 selecting a component from a set of candidates. In our proposal, we are adapting this model as a basis for quality measurement. It allows to express the semantics distance in terms of a functional suitability measure, which provides a better identification of the different COTS functionalities.

In section 2 of the paper, we introduce the component-based model for measurement (from [2]) (called here “component mapping diagram”) along with a motivating example. Then, section 3 presents a compact suite of measures – including functional suitability measures. Finally, section 4 addresses conclusions and topics for further research.

## 2 A Component-Based Model for Measurement

Component architectures divide software components into requiring and providing: some software components can register the services they provide, while other components can subscribe to and consume these services. Components are plugged into a software architecture that connects participating components and enforces interaction rules. The model in [2] supposes that there is an architectural definition of a system, whose behaviour has been depicted by scenarios or using an architecture description language (ADL).

The system can be extended or instantiated through the use of some component type. Due 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 behaviour specified by $S_C$, as shown in Figure 1 (from [2]).

The process of composing a component K with A is an act of interface and semantic mapping. In this work, only the semantic mapping will be addressed. We focus on incompatibilities derived from behavioural differences between the specification of a component $K_i$ ($S_{Ki}$) 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 in-

clude some of those specified by $S_C$. An additional necessary condition is that K provides at least the functional mapping between the domains as specified by $S_C$.

A typical situation for inconsistency in the functional mappings between $S_K$ and $S_C$ is illustrated by [2] in Figure 2, 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 and output 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.

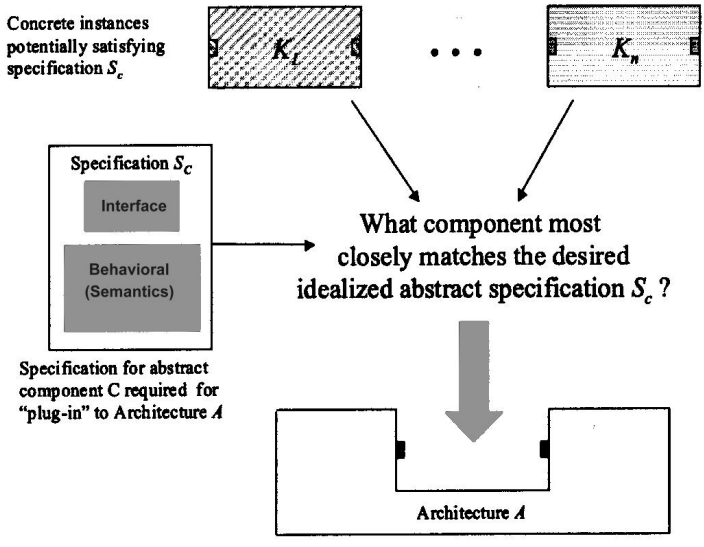


**Fig. 1.** Instantiation of an abstract component specification

### 2.1 A Motivating Example: E-payment Components

*Authorisation* and *Capture* are the two main stages in the processing of a card payment over the Internet. Authorisation 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 authorisation request if the retailer can guarantee a specific delivery time. Otherwise the capture will happen when the goods are shipped.

We suppose the existence of some scenarios describing the two main stages, which represent here a credit card (CCard) payment system. The scenarios will provide an abstract specification of the input and output domains of $S_C$ that might be composed of:

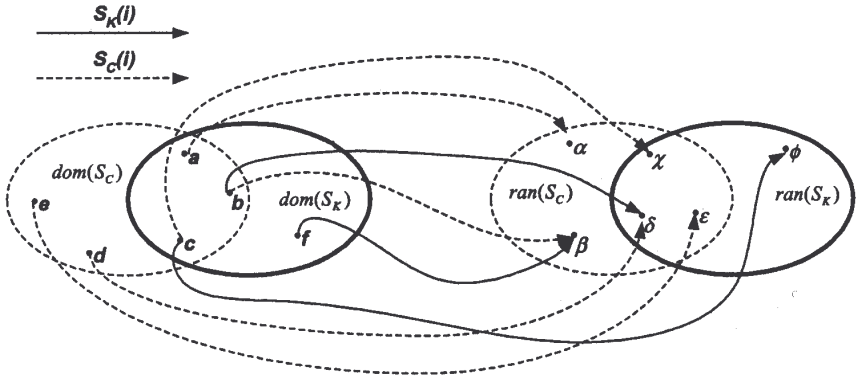


**Fig. 2.** Functional mappings of $S_C$ and $S_K$

- 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 → AOD; CID → COD}.

Suppose we pre-select two components to be evaluated, namely $K_1$ and $K_2$ respectively. The specification mapping, shown in Figure 3, reveals some inconsistencies that should be analysed. Firstly, 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. Secondly, the input domain of the component $K_2$ includes more values than the required by $S_C$, however the mapping satisfies the required functionality. We should note that there is another functionality provided by $K_2$, which might inject harmful effects to the final composition. Thus a deeper analysis based on previously defined scenarios should be carried out.

## 3 A Measurement Suite for Functional Suitability

For the measure definitions, we assume a conceptual model with universe of scenarios $\Sigma$, an abstract specification of a component $X$, a set of components $K$ relevant to $X$ and called candidate solution $\Sigma O$, a set of pre-selected components from $\Sigma O$, called solution $\Sigma N$, and a mapping component diagram $MX\Delta$. In this diagram, $S_C(i)$ represents the map associated to the input value $i$ defined in the domain of $S_C$. This map should provide a valid value on the output domain of $S_C$, i.e. there is no empty maps or invalid associations. A similar assumption is made on the mappings of $S_K$.

Let's briefly clarify the concepts associated to $\Sigma O$ and $\Sigma N$. Candidate components, selected from different sources for evaluation, constitute the members of the set $\Sigma O$. It could be the case that one of these members does not offer any functionality re-

quired by *X*. 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 get the requirements of *X* is called here *ΣN*.

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. For the credit card example, input/output data are represented by {AID, CID}, {AOD, COD} respectively.

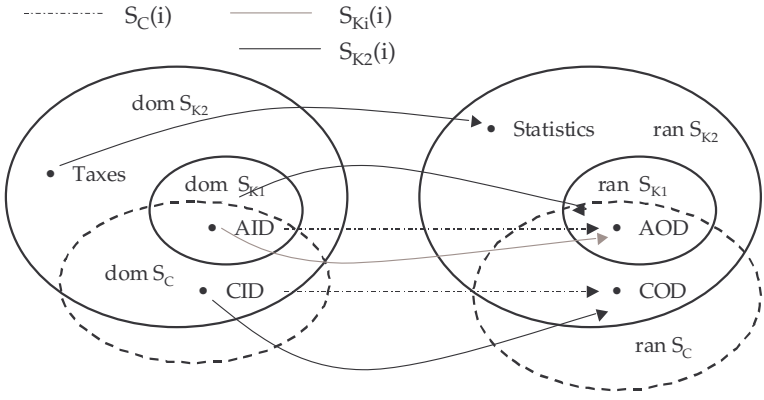


**Fig. 3.** Functional mappings of $S_C$ and $S_{K1}/S_{K2}$

### 3.1 Domain Compatibility Measures

The importance of defining domain compatibility measures comes from the importance of simplifying the COTS selection process. When analysing components, it might be the case that the data required by a concrete component *K* does not semantically match with the data required by its abstract specification *X*. Then, after determining the input/output compatibility, the analysis of the component *K* might stop (depending on the incompatibility detected), avoiding higher selection effort investments.

Table 1 lists the proposed measures for detecting input domain incompatibilities. 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 *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 *X*. In this case, the second group of metrics evaluates the domain compatibility of all components that constitutes the candidate solution *ΣO*, as we previously defined.

**Table 1.** Description of the ID-Compatibility measures

| Measure Id. | Description |
|---|---|
| Component-level | |
| $CID_C$ Compatible Input Domain | The number of input data provided by $S_K$ and required by $S_C$ in the scenario $\mathcal{S}$. |
| $MID_C$ Missed Input Domain | The number of input data required by $S_C$ in the scenario $\mathcal{S}$ and NOT provided by $S_K$. |
| $AID_C$ Added Input Domain | The number of input data NOT required by $S_C$ in the scenario $\mathcal{S}$ and provided by $S_K$. |
| $CC_{ID}$ Component Contribution | Percentage in which a component contributes to get the input data required by $S_C$ in the scenario $\mathcal{S}$. |
| Solution-level | |
| $SN_{ID}$ Candidate Solution | The number of components that contribute with compatible input data to potentially get the requirements of $S_C$ in the scenario $\mathcal{S}$. |
| $CID_S$ Compatible Input Domain | The number of input data provided by $\mathcal{SN}$ and required by $S_C$ in the scenario $\mathcal{S}$. |
| $MID_S$ Missed Input Domain | The number of input data required by $S_C$ in the scenario $\mathcal{S}$ and NOT provided by $\mathcal{SN}$. |
| $AID_S$ Added Input Domain | The number of input data NOT required by $S_C$ in the scenario $\mathcal{S}$ and provided by $\mathcal{SN}$. |
| $SC_{ID}$ Solution Contribution | Percentage in which a solution contributes to get the input data required by $S_C$ in the scenario $\mathcal{S}$. |

The input domain measure definitions are shown in Table 2. Similarly, a compatibility analysis of the output domain should be done, considering the data provided by the component *K* and using a similar suite of measures.

To clarify the reading, we should note that the expression $CID_C(K,C) \geq 1$ has been included to reduce the candidates for evaluation. This expression limits the analysis of missed and added inputs to those components that have already showed having at least a compatible input data. We should also remark the importance of determining semantics incompatibilities through the use of scenario specifications, even thought the scenario $\Sigma$ is not explicitly included into our measure definitions. This is due to the fact that we consider the definition of metrics as a process included into a broader measurement process, which defines some activities for setting the measurement context – such as defining scenario specifications or identifying stakeholders [6].

Now, let's calculate the input domain compatibility measures for our credit card example. The input domain of the abstract specification $S_c$ is {AID, CID}, and the input domains for *K1* and *K2* are {AID} and {AID, CID} respectively.

The following values of the measures:

$CID_C(K1) = 1$; $MID_C(K1) = 1$; $AID_C(K1) = 0$; and $CC_{ID}(K1) = 0.5$
$CID_C(K2) = 2$; $MID_C(K2) = 0$; $AID_C(K2) = 1$; and $CC_{ID}(K2) = 1$

show that the component *K*1 is a candidate to be discharged due to the existence of another component, *K*2, that is completely input compatible ($CC_{ID}(K2) = 1$). Hence, solution-level metrics are not calculated since our candidate solution has only one

component. Then, our functional suitability measurement will continue only considering *K*2 for analysis.

**Table 2.** ID-Compatibility measures

| Measurement element | Measure Id. | Measure Definition |
|---|---|---|
| Component $\mathcal{K}$ and component $\mathcal{C}$ | $CID_C$ | $\#\{df : data \mid df \in (domS_C \cap domS_K)\}$ |
| Component $\mathcal{K}$ and component $\mathcal{C}$ | $MID_C$ | $\#\{df : data \mid df \in (domS_C \setminus domS_K) \wedge CID_C(K,C) \geq 1\}$ |
| Component $\mathcal{K}$ and component $\mathcal{C}$ | $AID_C$ | $\#\{df : data \mid df \in (domS_K \setminus domS_C) \wedge CID_C(K,C) \geq 1\}$ |
| $CID_C$ and component $\mathcal{C}$ | $CC_{ID}$ | $\frac{CID_C}{\#(domS_c)}$ |
| Solution $\mathcal{SO}$ and component $\mathcal{C}$ | $SN_{ID}$ | $\#\{S_K \mid (CID_C(K,C) \geq 1) \forall K \in SO\}$ |
| Solution $\mathcal{SN}$ and component $\mathcal{C}$ | $CID_S$ | $\#\{d : data \mid df \in domS_C \cap \bigcup(domS_K) \forall K \in SN\}$ |
| Solution $\mathcal{SN}$ and component $\mathcal{C}$ | $MID_S$ | $\#\{d : data \mid df \in domS_C \setminus \bigcup(domS_K) \forall K \in SN\}$ |
| Solution $\mathcal{SN}$ and component $\mathcal{C}$ | $AID_S$ | $\#\{d : data \mid df \in (\bigcup domS_K) \forall K \in SN \setminus domS_C\}$ |
| $CID_S$ and component $\mathcal{C}$ | $SC_{ID}$ | $\frac{CID_S}{\#(domS_c)}$ |

### 3.2 Functional Suitability Measures

The 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. For example, the component *K*2 is full domain compatible, but some of the domain values might produce different functionalities from the required by the abstract specification of *X*, i.e. the input AID might produc-COD or any other output value. 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. Table 3 lists our suite of functional suitability measures, which are again classified into two groups: component-level measures and solution-level measures. A more formal definition of the measures is shown in Table 4[1].

[1] 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.

**Table 3.** 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 $\mathcal{S}$. |
| $MF_C$ Missed Functionality | The number of functional mappings required by $S_C$ in the scenario $\mathcal{S}$ and NOT provided by $S_K$. |
| $AF_C$ Added Functionality | The number of functional mappings NOT required by $S_C$ in the scenario $\mathcal{S}$ 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 $\mathcal{S}$. |
| 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 $\mathcal{S}$. |
| $CF_S$ Compatible Functionality | The number of functional mappings provided by $\mathcal{SN}$ and required by $S_C$ in the scenario $\mathcal{S}$. |
| $MF_S$ Missed Functionality | The number of functional mappings required by $S_C$ in the scenario $\mathcal{S}$ and NOT provided by $\mathcal{SN}$. |
| $AF_S$ Added Functionality | The number of functional mappings NOT required by $S_C$ in the scenario $\mathcal{S}$ and provided by $\mathcal{SN}$. |
| $SC_F$ Solution Contribution | Percentage in which a solution contributes to get the functionality required by $S_C$ in the scenario $\mathcal{S}$. |

Now, let's calculate the functional suitability measures for our credit card example. The functional mapping of the abstract specification $S_C$ is {AID → AOD; CID → COD}, and the functional mapping for *K*2 is {AID → AOD; CID → COD; *Taxes* → *Statistics*}. Then, the component-level measure results show the following values:

$CF_C(K2) = 2$; $MF_C(K2) = 0$; $AF_C(K2) = 1$; and $CC_F(K2) = 1$.

These values indicate that the component *K*2 is a candidate to be accepted for more evaluation, i.e. the component is functionally suitable but there is one added functionality that could inject harmful side effects into the final composition. 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 [5], analysis of vendor viability [3], and so forth. Our set of measures are only providing a way of identifying suitable components from a functional point of view. Measuring the other aspects is still a remaining issue. Another interesting discussion will be on analysing 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 [4] remarks.

**Table 4.** Functional Suitability measures

| Measurement element | Measure Id. | Measure Definition |
|---|---|---|
| Component $\mathcal{K}$ and component $\mathcal{C}$ | $CF_C$ | $\#\{df : data \mid df \in (domS_C \cap domS_K) \wedge S_K(df) = S_C(df)\}$ |
| Component $\mathcal{K}$ and component $\mathcal{C}$ | $MF_C$ | $\#\{df : data \mid CF_C(K,C) \geq 1 \wedge df \in domS_C \wedge (S_C(df) \neq S_K(df) \vee df \notin domS_K)\}$ |
| Component $\mathcal{K}$ and component $\mathcal{C}$ | $AF_C$ | $\#\{df : data \mid CF_C(K,C) \geq 1 \wedge df \in domS_K \wedge (S_C(df) \neq S_K(df) \vee df \notin domS_C)\}$ |
| $CF_C$ and component $\mathcal{C}$ | $CC_F$ | $\frac{CF_C}{\#(S_C)}$ |
| Solution $\mathcal{SO}$ and component $\mathcal{C}$ | $SN_F$ | $\#\{S_K \mid (CF_C(K,C) \geq 1)_{\forall K \in SO}\}$ |
| Solution $\mathcal{SN}$ and component $\mathcal{C}$ | $CF_S$ | $\#\{df : data \mid df \in (domS_C \cap \bigcup(domS_K)_{\forall K \in SN}) \wedge (S_K(df) = S_C(df))_{\exists K \in SN}\}$ |
| Solution $\mathcal{SN}$ and component $\mathcal{C}$ | $MF_S$ | $\#\{df : data \mid df \in (domS_C \setminus \bigcup(domS_K)_{\forall K \in SN}) \vee (df \in domS_C \cap \bigcup(domS_K)_{\forall K \in SN} \wedge (S_C(df) \neq S_K(df))_{\forall K \in SN})\}$ |
| Solution $\mathcal{SN}$ and component $\mathcal{C}$ | $AF_S$ | $\#\{df : data \mid df \in \bigcup(domS_K)_{\forall K \in SN} \setminus domS_C \wedge (S_K(df))_{\exists K \in SN})\}$ |
| $CF_S$ and component $\mathcal{C}$ | $SC_F$ | $\frac{CF_S}{\#(S_C)}$ |

Finally, our measures on functional suitability could provide a more precise indicator when calculating the maintenance equilibrium value as introduced in [1]. The number of components in the solution ($SN_F$) should be minimised and the contribution of functionality ($SC_F$) should be maximised to satisfy the CBS Functional Density Rule of Thumb: "*Maximise the amount of functionality in your system provided by COTS components but using as few COTS components as possible*" [1].

## 4 Conclusions and Future Work

We have presented a preliminary suite of measures for determining the functional suitability of a component-based solution. However, our measures are based on functional direct mappings, i.e. there is no semantic adaptation between the outputs provided by a component *K* and the required functionality in *X*. Therefore, we are extending the suite presented here to quantify the semantic adaptation providing an integral suite of measures.

Finally, all the measures need to be empirically validated, so much research must still be done to demonstrate the applicability of our proposal.

**Acknowledgments.** This work is partially supported by the CyTED (Ciencia y Tecnología para el Desarrollo) project VII-J-RITOS2, by the UNComa project 04/E048, 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

1. C. Abts. COTS-Based Systems (CBS) Functional density - A Heuristic for Better CBS Design. In *Proceedings of the First International Conference on COTS-Based Software Systems*, Springer Verlag LNCS 2255, pages 1–9, 2002.
2. R. Alexander and M. Blackburn. Component Assessment Using Specification-Based Analysis and Testing. *Technical Report SPC-98095-CMC, Software Productivity Consortium*, 1999.
3. K. Ballurio, B. Scalzo, and L. Rose. Risk Reduction in COTS Software Selection with BASIS. In *Proceedings of the First International Conference on COTS-Based Software Systems*, Springer Verlag LNCS 2255, pages 31–43, 2002.
4. B. Bertoa, J. Troya, and A. Vallecillo. A Survey on the Quality Information Provided by Software Component Vendors. In *Proceedings of the ECOOP QAOOSE Workshop,* 2003.
5. B. Bertoa and A. Vallecillo. Quality Attributes for COTS Components. In *Proceedings of the ECOOP QAOOSE Workshop*, 2002.
6. A. Cechich and M. Piattini. Defining Stability for Component Integration Assessment. In *Proceedings of the Fifth International Conference on Enterprise Information Systems*, pages 251–256, 2003.
7. J. Dolado. A Validation of the Component-Based Method for Software Size Estimation. *IEEE Transactions on Software Engineering,* 26(10):1006–1021, 2000.
8. W. Hansen. A Generic Process and Terminology for Evaluating COTS Software –The QESTA Process. http://www.sei.cmu.edu/staff/wjh/Qesta.html.
9. L. Holmes. Evaluating COTS Using Function Fit Analysis. Q/P Management Group, INC - http://www.qpmg.com.
10. ISO International Standard ISO/IEC 9126. ISO/IEC 9126 - Information technology - Software product evaluation - Quality characteristics and guidelines for their use, 2001.
11. S. Sedigh-Ali, A. Ghafoor, and R. Paul. Software Engineering Metrics for COTS-Based Systems. *IEEE Computer Magazine*, pages 44–50, May 2001.