Using Quality Models for Assessing COTS Selection[¥]

Pere Botella, Xavier Burgués, Juan P. Carvallo[‡], Xavier Franch, Carme Quer

Universitat Politècnica de Catalunya (UPC)
c/ Jordi Girona 1-3 (Campus Nord, C6) E-08034 Barcelona (Catalunya, Spain)
{botella, diafebus, carvallo, franch, cquer}@lsi.upc.es

Abstract. We present in this paper a framework embracing different aspects involved in COTS component selection that influence the success of this activity. Playing a crucial role in this framework appears the concept of quality model, aimed at structuring the description of the quality of COTS components. We propose a methodology for building quality models based on the ISO/IEC 9126-1 standard which allows to create hierarchies of models appropriate for categories and domains of COTS components, and also for particular contexts of COTS selection activities. Such quality models facilitate the expression and refinement of quality requirements during COTS selection. We present also a formal notation for expressing these quality models, the quality requirements and the product descriptions themselves; the notation supports model analysis and makes feasible tool support during COTS selection. Last, we enumerate at the conclusions some issues matter of current and future research.

1. Introduction

The growing importance of *commercial-off-the-shelf software components* (hereafter *COTS components* or simply *COTS*) requires adapting some software engineering practices, such as requirements elicitation to this emergent framework. Also some specific activities arise, among which COTS selection [1] plays a prominent role.

COTS selection poses some questions to be addressed such as:

- How COTS components can be arranged in categories and domains for knowing which is the current state of the COTS market?
- How COTS components from a given domain are described, to make feasible their comparison when selection is required?
- How features of COTS components may be reconciled with requirements on them?
- Is it possible and realistic to describe COTS components and requirements in a structured and even formal way?

This paper addresses mainly these questions, being aware that many others are hidden behind the curtain. More precisely: we put together some results we have obtained in previous work; we present new advances in some of these results; we identify some

¥ This work is partially supported by the Spanish research program CICYT under contract TIC2001-2165.

‡ Juan P. Carvallo's work has been supported by an AECI grant.

key factors in the various activities taking place in our framework; and we outline some future research. The main ideas in this paper may be summarised as follows:

- We recognise the need of building a description of the COTS market to be able to address a particular selection activity to the right market segment. We organise this description as a taxonomy enclosing different types of concepts.
- We propose quality models [2, 3] as the central notion to articulate COTS components selection, more precisely ISO/IEC-9126-1-based quality models. Quality models provide a means for defining quality characteristics of components and metrics[1]. We have linked quality models with the taxonomy above.
- We show the use of quality models for the description of COTS components and the formulation and refinement of quality requirements.
- We aim at expressing quality models in a formal way, using a structured notation named NoFun. This notation allows to express models, descriptions of COTS components with respect to these models, and requirements over them. NoFun catches the overall structure of ISO/IEC-based quality models ameliorating then the cost of the formalisation process.

2. Building a taxonomy for COTS domains

The market of COTS components is huge and highly dynamic. On the one hand, new types of COTS components, i.e. *COTS domains*, appear day by day (e.g., the domain of XML technologies). On the other hand, new COTS components embrace often capabilities from more than one type, especially when they evolve through the years, making more difficult their analysis; this is the case of e-mail client packages, which often offer also functionalities for chatting or scheduling meetings, for instance.

For these reasons, we advocate than improving the effectiveness and confidence of COTS selection requires:

- Having a taxonomy for arranging COTS domains. The existence of such taxonomy provides a framework for the whole selection process and structures knowledge on the field. The intermediate nodes of the taxonomy stand for general *COTS categories*: they are just a classification means, not real COTS domains. Fig. 1 presents an excerpt of how this taxonomy may look like. It shows that as many levels as needed may be introduced to catch similarities in the right point.
- Identifying the features that characterise each of these COTS categories and domains. These features capture the similarities of all COTS components belonging to the same COTS domain and also those of all COTS domains belonging to the same COTS category. Features are inherited down the hierarchy.
- Making explicit the relationships among COTS domains. We propose using a i* SD-model [4] to visualise these relationships; if enough knowledge exists, parts of the SD-model may be refined into SR-ones. Fig. 2 shows an i* SD-model for making explicit a few relationships among the mail server and mail client COTS domains, and also the meeting scheduler one. Domains are modelled using the i*

[1] We do not consider criteria other than quality in COTS selection, being aware that some of them that can be at least as important as quality, namely cost, confidence on the supplier, etc.

notion of intentional agent, while relationships take the form of dependencies. It becomes clear that scheduling meetings requires the ability of sending messages (goal dependency) and access to address books (resource dependency); also the mail client relies on the mail server on sending those messages. The i* SD-model shows that a COTS meeting scheduler not providing mailing facilities, requires a mail client COTS product to exist or to be also acquired. This kind of multiple selection has been addressed in [5].

- Classifying COTS components as belonging to one or more of these domains. This classification is the first step in determining which is the quality model bound to the COTS components, as we will do in the next sections.

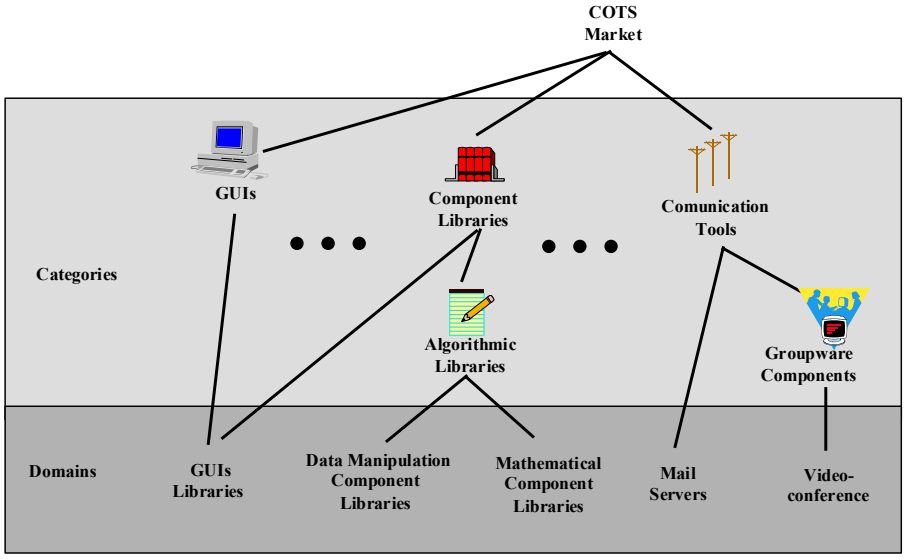

Fig. 1. An excerpt of the ongoing taxonomy for COTS market**.**

Key success factors in defining this taxonomy are:

- Define the right COTS categories and their proper decomposition. An appropriate number of categories and levels of the hierarchy is needed for presenting a good trade-off between knowledge structure and taxonomy management. Also, future evolution of the hierarchy must not be compromised by too early decisions.
- Define the right COTS domains. The granularity must be fine-grained enough to avoid failure of COTS components classification and wide-grained enough to avoid proliferation of artificial domains.
- Focus on functional dependencies among COTS domains, not on non-functional ones. If a functional dependency exists from one COTS component to another, for sure other non-functional ones exist, but we feel it is not necessary to reflect them at this stage. For instance, if the meeting scheduler needs a mail client to send messages, this implies that the reliability of the scheduler depends partly on the reliability of the mail client. This decision avoids proliferation of dependencies in the i* SD-model that are not useful in this context.

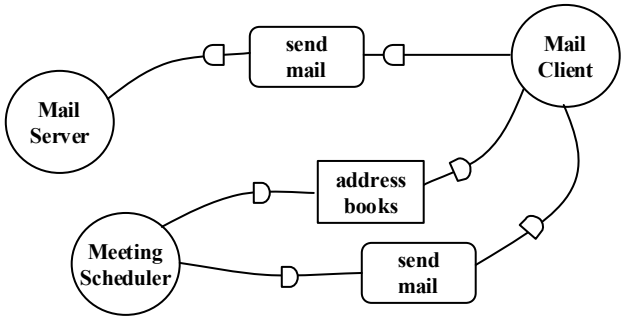

Fig. 2. An excerpt of the i* SD-model involving the meeting scheduler COTS domain

- Identify the appropriate set of features for each COTS category and domain. This set should be kept minimal to avoid having useless or meaningless features at any place of the hierarchy.

COTS component classification is currently a focus of interest in many contexts, both purely academic and commercial. Concerning academic proposals, [6] identifies some relevant criteria for building a COTS market classification. This approach is more general than ours with respect to criteria, because they are not restricted to quality; but on the other hand, classification is a goal by itself, while in our proposal is a starting point for quality model definition. In the commercial side, many COTS markets in the web make intensive use of classification, although criteria is not always clear.

3. Using quality models for describing COTS domains

One of the key success factors for building the COTS taxonomy has been mentioned to be identification of the right features. Features have to be with different kind of factors, such as managerial, political and of course quality characteristics. In the rest of the paper, we are going to focus on this specific kind of features.

There are a lot of approaches for specifying quality features. We propose the use of *quality models* as the framework for arranging these quality features and for defining their metrics. There are also some proposals for defining quality models [7, 8]. From our point of view, the main requirements over these proposals are:

- They should just fix some high-level quality concepts. This is a crucial point, because quality models may dramatically differ from one domain to another.
- They should allow creating taxonomies of quality features, which is essential in order to build structured quality models and which also facilitates the integration of quality models with the COTS taxonomy.
- These hierarchies should allow overlapping, since quality features may contribute to others in different ways.
- They should be widespread. This discards ad-hoc proposals that may look appealing and promising but that are not currently used by the software engineering community.

One of the obvious candidates fulfilling these requirements is the ISO/IEC 9126-1 quality standard [3]. As an additional point supporting this choice, just to mention that this standard is integrated with others of interest, namely the whole 9126 family (currently not yet delivered, although draft versions can be obtained), the 14598 for software product quality and evaluation, 12207 for software life cycle, 15504 for process assessment and of course ISO 9001 for quality assurance processes.

An ISO/IEC 9126-1 quality model is defined by means of general *characteristics* of software, which are further refined into *subcharacteristics*, which in turn are decomposed into *attributes*, yielding to a multilevel hierarchy; intermediate hierarchies of subcharacteristics and attributes may arise. At the bottom of the hierarchy appear the measurable software attributes, whose values are computed by using some *metric*.

4. Types of quality models

Being effectiveness one of our aims, it could be argued that building quality models is a time-consuming activity. Therefore, improvements in this direction are welcome.

Reusability of quality models among different COTS domains can be helpful for this objective. We have observed throughout our experiences that some quality entities appear over and over. This observation has lead us to the definition of five different types of quality models, which roughly correspond to the levels in the taxonomy presented in section 2. The recognition of COTS domains and categories improves reusability: once a new COTS domain has been identified, its quality model can be constructed by inheriting the features of the quality models for those COTS categories in the hierarchy which it belongs to. Since then, any quality model for a particular selection process may reuse the quality model of the corresponding COTS domain. The types of quality models are presented next.

4.1 Context-free quality model

The ISO/IEC 9126-1 quality standard is intentionally vague, for the sake of generality. In the framework of COTS components, some of the proposed subcharacteristics may be further decomposed, identifying other subcharacteristics and even attributes, resulting then in a new *context-free* quality model that will be used as starting point of any other specific quality model.

A typical example appears in the *Suitability* subcharacteristic. Successful COTS components tend to bind applications that were not originally related to them. This is particularly true if one considers that product suppliers try to include some features to make their products different from the others. These added applications are not usually shipped within the original COTS components; they are offered separately, as extensions of the original one. But in many cases, they are referenced as a constitutive part of the functions provided by the component. As a result, we may split the *Suitability* characteristic into two, *Basic Suitability* and *Added Suitability*, keeping track of both of them inside the model but in a clearly separated way.

4.2 Category quality model

These models characterise all the domains in one category. There are two possible starting points to obtain a category quality model:

- Departing from the context-free quality model, through modifications, usually additions. One example is the *Encryption Algorithm* attribute, which would be added to the *Security* subcharacteristic in the *Communication Tools* (see fig. 1) quality model. Elimination of subcharacteristics also occurs sometimes. For instance, the subcharacteristic *Attractiveness* defined in the standard (use of colour, graphical appearance, etc.) appears in the context-free quality model, but it does not apply in domains that are purely pieces of software to be integrated in a system.
- Consolidating the quality models of the more general categories in the taxonomy, and then modifying the result by adding particularities of the specific category. This way to obtain category quality models becomes useful when new categories appear in the COTS market that can be classified as subcategories of more general ones. This could be the case of a *New Language Compiler* category that would be subcategory of the *Compiler* and *Development Tools* categories.

4.3 Domain quality model

These models characterise types of COTS components. This is the type of quality model we are mostly interested in, since they give an exhaustive and structured description of COTS components in a domain to be used widespread and to serve as a framework in which particular components may be evaluated and compared to user requirements during a selection process. Domain quality models will be specially appealing in the selection of components in a COTS domain that satisfy two conditions: they are needed by a huge number of companies and there are lots of COTS available in the market. Context-free and category quality models are mainly means to obtain domain quality models. As stated above, category quality models will facilitate the construction of new domain quality models, since they will avoid to construct these models from the scratch [9]. Examples of COTS domains are *Mail Servers*, *Videoconference Tools, Mathematical Component Libraries*, etc. (see fig 1).

4.4 Organisation-type quality model

Domain quality models can still be specialised depending on the organisation-type for which the selection process will be done. For instance, in mail servers, it would be possible to distinguish among quality models for ISP providers, small organisations and large organisations. Some modifications will be necessary in order to adequate the domain quality models to the type of organisation. For instance, the metrics for a certain attribute may be different depending on this type. In the selection of mail servers, a small organisation can be just interested in the existence of an encryption algorithm, whilst a larger one will be interested in which algorithms are provided.

4.5 Final quality model

These models will be constructed as a previous step of a particular selection process, starting from the corresponding organisation-type quality model and taking into account the particularities of the organisation context. Some attributes in a quality model cannot be defined without taking into account one specific organisation. One example could be a *Quality of Interface* attribute, since its definition and its metrics can vary depending on the concrete organisation for which the procurement is done.

5. A methodology for building domain quality models

Next we outline a methodology for building ISO/IEC 9126-1 quality models. We focus on domain quality models, which are more generic than the other four ones. Details may be read in [9, 10], although some differences exist, remarkably the distinction between different types of models presented in the last section.

Step 0. Analysing the domain
The domain of interest has to be carefully examined and described. With respect to the first point, experts in the domain must join the quality team. Concerning the second point, formal models can be build to keep track of all the relevant concepts.

A first model for the domain is the i* model we have suggested to build in section 2 for making explicit dependencies among domains. The agents appearing there may be further decomposed (into hidden agents) for making explicit the most relevant functionalities of the component (e.g., the mail server may be decomposed into folder manager, message manager and account agents, among others). Together with this i* model, a conceptual model (e.g., a UML class diagram) helps on making more explicit the structure of the resources involved in the domain.

Step 1. Classifying the domain and building the initial quality model
The domain has to be integrated into the COTS taxonomy, which means identifying the category or categories which it belongs to. As a result, an initial quality model is obtained by putting together the quality entities and metrics contained in the inherited models. Of course, inconsistencies must be detected.

Step 2. Determining the first-level quality subcharacteristics
The decomposition of characteristics into subcharacteristics appearing in the departing model is quite reasonable and should be used unless very good reasons for not doing so come out during domain analysis. In these cases, the quality team may add new subcharacteristics specific to the domain, refine the definition of some existing ones, or even eliminate some. For instance, in the domain of data structure libraries, the *Time Behaviour* subcharacteristic may be refined as "execution time of the methods provided by the classes inside the library".

Step 3. Defining a hierarchy of subcharacteristics
Many subcharacteristics may be further decomposed with respect to some factors, yielding to a hierarchy. For instance, in some domains as the one for e-learning tools, the attributes categorised under the *Operability* subcharacteristic of *Usability* may be seen from two different points of view: the general user and the administrator. So, it makes sense to decompose this subcharacteristic into two, one for each type of user.

Step 4. Decomposing subcharacteristics into attributes
Quality subcharacteristics provide a comprehensible abstract view of the quality model. But next it is necessary to go into the details, by decomposing these abstract concepts into more concrete ones, the quality attributes. An attribute keeps track of a particular observable feature of the packages in the domain. For example, attributes in the *Learnability* subcharacteristic may include *Quality of Graphical Interface* of the product, *Number of Languages Supported* and *Quality of Available Documentation*.

Step 5. Decomposing derived attributes into basic ones
Some of the attributes emerging in step 4 may be directly measurable given a particular product (e.g., *Number of Languages Supported*) but others may be still abstract enough to require further decomposition. This is the case of the *Quality of Graphical Interface* attribute mentioned above; quality may depend in various factors, as user-friendness, depth of the longest path in a browsing process, types of interface supported, etc. Thus, we distinguish between *derived* and *basic attributes*. Derived attributes should be decomposed until they are expressed in terms of basic ones.

Derived attributes may be completely defined in terms of their components or not. In some situations, giving a concrete definition of the quality interface attribute could be considered harmful, because it would force to use always the same definition without considering the requirements of a particular context [8]. Sometimes requirements may give more importance to the user-friendness factor (e.g., for non-skilled users), sometimes to its type (for interoperability purposes) and so on. In this case, the definition of the derived attribute is postponed. We call the first case of derived attributes *context-free*, while the second ones are *context-dependent*.

Step 6. Stating relationships between quality entities
To obtain a real complete quality model, relationships between quality entities must also be explicitly stated. The model becomes more exhaustive and as an additional benefit, implications of quality user requirements may become clearer.

We may identify various types of relationships among quality entities: collaboration, damage and neutral dependency. Elaborate types and also intensities of these relationships may be built, as done in [11, 12].

Step 7. Determining metrics for attributes
Not only the attributes must be identified, but also metrics for all the basic attributes must be selected, as well as metrics for those derived context-free attributes. The standard ISO/IEC 9126-2 can be used for this purpose as well as metrics theory [13].

Metrics for basic attributes are quantitative. Derived context-free attributes may be either quantitative or qualitative, with explicit formula computing their value from their component attributes.

Some attributes require an elaborated representation, yielding to *structured* metrics. Examples are sets (e.g., set of labels for the languages supported by the interface) and functions. Functions are especially useful for attributes that depend on the underlying platform. For instance, many attributes related to the time behaviour subcharacteristic may fall into this category.

Step 8. Identifying requirement patterns for quality entities
Once the model is complete, we may go a step further by trying to facilitate its use during COTS component selection. A particular way for bridging the gap among definition and use of the quality model is to identify some typical requirements on the quality entities of the model. During the selection process, this catalogue of requirements may help the elicitation phase. Usual types of requirements are: maximising/minimising values of attributes (e.g., the mean time between failures should be kept minimum as possible), satisfying a certain value (e.g., messages must be presented in Spanish), etc.

Step 9. Formalising the quality model
The last step identified in the methodology for building quality models requires formalisation of the entities and metrics that appear therein. The goal of formalisation is twofold: on the one hand, some ambiguities, inconsistencies and incompleteness will surely arise, as it is the usual case in any formalisation process; on the other hand, formal descriptions of quality models are a requirement for providing some tool support to the COTS selection problem. See section 7 for details.

Some key factors for obtaining quality models for COTS domains
- Experts of the field must participate in the quality team. Be abstract! Keep in mind that the goal is to define a general framework for many applications of the same brand, not one for a particular product.
- Skilled software engineers are necessary to properly manage conceptual and agent-oriented models.
- Use a vocabulary. One of the most endangering points is the lack of standard terminology in the components of the domain. The same concepts are named different by different vendors or even worse, the same name may denote different concepts in different packages.

6. Package and requirement descriptions

One could wonder whether or not the construction of a quality model is a too complex and time-consuming activity, in spite of the use of hierarchies for supporting reusability. The answer is the overhead required for building quality models is compensated by their use in more than one single selection process:

- Quality models provide a general framework to get uniform descriptions of the COTS in the domain (see fig. 3). Analysis of these COTS is favoured, improving the reliability of the selection process.
- Quality requirements can be formulated in a structured manner in terms of the quality concepts appearing in the model (see fig. 3). This process may help to discover some ambiguities and incompleteness and, once solved, the resulting requirements can be more easily compared with the COTS descriptions.

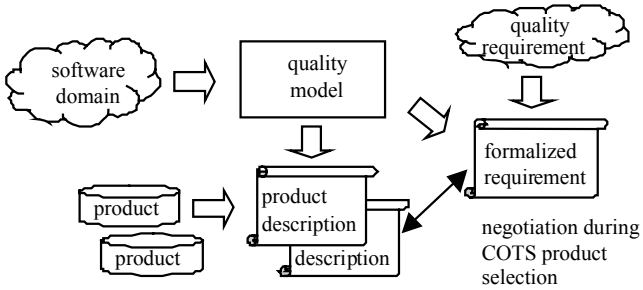

Fig. 3. Using quality models in the COTS selection process

Concerning the definition of quality requirements, we show in fig. 4 some requirements in the domain of mail servers components [10]. They illustrate some situations that are common when requirements are analysed using quality models.

Req.	Requirement Description
1	Spanish language support
2	Support for the most commonly used certification standard
3	Support for accessing the server from other applications
4	Mail delivery notifications, possibility of configuring parameters such as maximum number of delivery retries, and time between them
5	Transmission time less than 1 minute for messages without attachments. For those with attachments it should not exceed 5 minutes per megabyte

Fig. 4. Some sample requirements on the mail servers COTS domain

- Requirement 1. Directly mapped in terms of an attribute of the model.
- Requirement 2. The involved attribute is clear, but it requires mapping the expression "most commonly used certification standard" to the value "X.509".
- Requirement 3. A too general requirement: what does it mean “other applications”? Further interaction and perhaps negotiation to get a more detailed specification must be carried out to better classify it.
- Requirement 4. Requires or implies a mixture of functionalities, which may be supported by selecting several attributes. Further feedback may be required in order to better classify this kind of requirements.
- Requirement 5. Example of ill-formulated requirement. It is not accurate since the one-minute limit for messages without attachment could be unfeasible when large data is included inline (e.g., annual company reports). So we must reformulate it using the attributes *Average Response Time* and *Throughput*.

Among the key success factors in using the quality model in COTS selection we find:

- Don't trust anyone or anything unless you are really confident on the source of information. An evaluation of a product may not be done without installing it and performing some kind of hands-on experimentation.
- Select appropriate metrics for basic attributes, mostly quantitative. Rely on metrics theory [13] and keep a good balance among overspecification (time-consuming and strongly-theoretical metrics difficult to apply) and oversimplification (compromising confidence).
- When putting requirements in terms of the quality model, keep track of the original requirements. Traceability is the basis for understandability and evolution.
- Keep requirements as minimal as possible. One of the reported reasons for selection failure is over-constraining the space solution [14]. A helpful tactic here is the definition of requirements priorities, by assigning some score or qualitative label.

7. Formalising quality models and the requirements on them

The NoFun language [15, 16] is the notation we use for formalising quality models, component descriptions and quality requirements. NoFun basically encapsulates in modules the following kind of capabilities:

- Definition of quality models by declaring their characteristics, subcharacteristics and attributes, together with metrics for their context-free attributes (see 7.1). There are some structuring mechanisms that support hierarchies as defined in previous sections. This corresponds to the icon labelled "quality model" in fig. 3.
- Assignment of values to basic attributes for particular COTS components. This part of the language describes product quality in a formal way. This corresponds to the icon labelled "product description" in fig. 3.
- Statement of quality requirements and assessment criteria (see 7.2). Quality requirements are stated using operators over the quality entities, and they may be categorised depending on their importance. This corresponds to the icon labelled "formalized requirements" in fig. 3.

7.1 Quality models in NoFun

The next three figures introduce some elements to define a particular subcharacteristic for the ERP systems domain. Fig. 5 defines the subcharacteristic *Accuracy* for ERP systems, declared in terms of other subcharacteristics and attributes; modules defining them are imported. The subcharacteristic is considered to be context-dependent, so no definition is provided. Fig. 6 introduces some data domains that will be useful when building the quality model: the first one, *COMPANY_AREAS*, is particular of this domain while the other, *UPPER_ADEQUACY_SCALE*, is a general-purpose qualitative assessment metric. Last, fig. 7 shows one particular module defining a couple of attributes that were imported in fig. 5; the second attribute is derived and context-free, so its definition is given. Note the use of functions and sets.

```
subcharacteristic module ACCURACY for ERP_SYSTEM
   imports ORIENTATION, ...  // other modules required
   subcharacteristic Accuracy derived
          explanation Accuracy ISO/IEC subchar. bound to ERP domain
          in terms of AreaCoverage, ... derived
end ACCURACY for ERP_SYSTEM
```

Fig. 5. Definition of the accuracy subcharacteristic for ERP systems

```
domain module COMPANY_AREAS for ERP_SYSTEM
   explanation Areas or functions of a company
   domain CompanyAreas defined as Commercial, Logistics, Manufacturing,
                                  HumanResources, Accounting, Finances,
                                  Quality, Technical, Management Support
end COMPANY_AREAS for ERP_SYSTEM

domain module UPPER_ADEQUACY_SCALE
   explanation 5-value scale which penalises excessive coverage
   domain ordered UpperAdequacyScale
   defined as NonExistent, Low, Excessive, Medium, High
end UPPER_ADEQUACY_SCALE
```

Fig. 6. Definition of the some auxiliary domains used in specifying ERP systems

```
attribute module ORIENTATION for ERP_SYSTEM
   imports COMPANY_AREAS, UPPER_ADEQUACY_SCALE
   attribute AreaCoverage
      explanation Degree of coverage of company areas
      declared as function from CompanyAreas to UpperAdequacyScale
                                    default NonExistent
   attribute MainTarget derived
      explanation Company areas well-covered by an ERP product
      declared as set of CompanyAreas
      defined as set of a in CompanyAreas such that
                                      AreaCoverage(a) = High
end ORIENTATION for ERP_SYSTEM
```

Fig. 7. Definition of quality attributes for dealing with ERP systems orientation

7.2 Formalising quality requirements in NoFun

Quality requirements appear in two different contexts: as universal properties of quality entities, or as the criteria that rule COTS component selection. Fig. 8 shows an example of the first case: it is explicitly required that any ERP system must give support at least to one company area. Note that this kind of requirement is very restrictive and so its adequacy must be carefully analysed.

Fig. 9 outlines a possible scenario arising in a selection process for ERP systems performed in the context of the ACME organisation. The example is not self-contained; see [16] for a complete description. There are four requirements with different names and also priorities. The fourth requirement is decomposed into two.

We remark the use of some powerful constructs, such as: quantifiers in the first requirement; function and set operators in the first two requirements; expressions for maximising values in the repository of available COTS components, in the fourth requirement. Note than once the requirement module has been declared for a particular COTS domain (***for*** *ERP_SYSTEM*), references to quality entities such as adaptability or openness are implicitly referencing this domain.

```
requirement module ORIENTATION_PROPS on ORIENTATION for ERP_SYSTEM
   explanation Universal properties of ERP-orientation attributes
   definition
      notUseless: essential
         explanation ERP products must address at least one company area
          defined as    exists a in CompanyAreas
                            such that AreaCoverage(a) > NonExistent
end ORIENTATION_PROPS on ORIENTATION for ERP_SYSTEM
```

Fig. 8. Definition of a universal requirement for ERP systems orientation

```
requirement module ACME on FUNCTIONALITY for ERP_SYSTEM
      explanation ... // informal presentation
      definition
         miminalCoverage: essential
              explanation Selected ERP should cover all company areas
              concerns ORIENTATION
              defined as   for all a in CompanyAreas it holds that
                               AreaCoverage(a) > NonExistent
         importantAreas: important
              explanation Selected ERP should emphasize commercial,
                          logistic and management areas
              concerns ORIENTATION
              defined as {Commercial,Logistic,Management} in MainTarget
         adaptability: marginal
              explanation The company could adapt its structure to
                           the new software if necessary
              concerns ADAPTABILITY
              defined as Adaptability.CompanyAdaptability >= None
         openness: advisable
              explanation The selected ERP should be open both to add
              functionality and to interconnect with other software
              concerns OPENNESS
              decomposed as
                  minimalOpenness: advisable
                     defined as Openness.bespoke >= Strong and
                                Openness.COTS >= Strong
                  maximalOpenness: marginal
                      defined as
                          max(Openness.bespoke) and max(Openness.COTS)
end ACME on FUNCTIONALITY for ERP_SYSTEM
```

Fig. 9. An example of quality requirements for ERP system

8. Conclusions

In this paper we have presented a methodology aimed at building quality models based on the ISO/IEC 9126-1 quality standard. We have applied our methodology to some domains including mail servers, ERP systems, e-learning tools, data structure libraries, and others. Together with this central idea, we have remarked the need for creating a taxonomy of COTS categories and domains; we have presented a formalisation language for structuring the quality models; and we have pointed out the connection between the quality model and the statement of quality requirements.

Quality models are especially appealing in COTS domains that satisfy two conditions: they are needed by a huge number of companies and there are lots of COTS available in the market. In these domains, a quality model can lower the cost of the very selection process. Just consider the amount of repeated work that is done in the selection processes of different organisations.

For lack of space, we have not been able to include other lines of current and future research:

- Selection of COTS components has been usually studied isolated. However, there are two cases that do not fit in this situation. On the one hand, an organisation may need at the same time different components for covering different needs, components that will have to be selected more or less simultaneously and that should be integrated. We have proposed a methodology for carry out this kind of selection [17]. On the other hand, even when there is just one component of interest, it may depend on others' existence, which should have to be selected in their turn, in case they do not exist. We have modelled these dependencies among COTS products using the i* model [5].
- We have already commented that COTS components boundaries are sometimes not clear. In particular, one COTS component may embrace different domains, either totally or partially. For a given COTS-based system architecture, described in terms of COTS domains, there are different way to integrate COTS products depending on their coverage of these involved domains. Different combinations of COTS products may lead to different architectural properties regarding reliability, efficiency and so on. We also have explored this issue in [5].
- UML, as a standard notation to represent models of systems, can be used basically to model functional requirements, but there is a lack for the non-functional ones. We have explored in [18] the possibility of porting NoFun to UML by extending class diagrams using the stereotype mechanism and OCL. In [19] we show how the added non-functional features could be applied at different modelling levels: to the whole system, to a subsystem or to a particular module or class. We are now initiating an experiment, based on a real application, in order to explore the possibility of complement the method proposed in [20] (a process-oriented approach) with our NoFun-based extension of UML (product-oriented), using the idea from [11] that both approaches can be complementary.

9. References

[1] A. Finkelstein, G. Spanoudakis, M. Ryan. "Software Package Requirements and Procurement". Proceedings of the 8th IEEE International Workshop on Software Specification and Design (IWSSD), 1996.

[2] B. Kitchenham, S.L. Pfleeger. "Software Quality: the Elusive Target". IEEE Software, vol. 20, January 1996.

[3] ISO/IEC Standard 9126-1 Software Engineering – Product Quality – Part 1: Quality Model, June 2001.

[4] E. Yu. "Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering". Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (ISRE), 1997.

[5] X. Franch, N. Maiden. "Modeling Component Dependencies to Inform their Selection". Proceedings of the 2nd International Conference on COTS-Based Software Systems (ICCBSS), 2003.

[6] L. Jaccheri, M. Torchiano. "Classifying COTS products". Proceedings European Conference on Software Quality, 2002.

[7] R.G. Dromey. "Cornering the Chimera". IEEE Software, vol. 20, January 1996.

[8] J. Bøegh, S. Depanfilis, B. Kitchenham, A. Pasquini. "A Method for Software Quality Planning, Control, and Evaluation". IEEE Software, vol. 23, March 1999.

[9] X. Franch, J.P. Carvallo. "A Quality-Model-Based Approach for Describing and Evaluating Software Packages". Proceedings of the 10th Joint International Conference on Requirements Engineering (RE), 2002.

[10] J.P. Carvallo, X. Franch, C. Quer. "Defining a Quality Model for Mail Servers". Proceedings of the 2nd International Conference on COTS-Based Software Systems (ICCBSS), 2003.

[11] L. Chung, B. Nixon, E. Yu, J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.

[12] H. Kaiya, H. Horai, M. Saeki. "AGORA: Attributed Goal-Oriented Requirements Analysis Method". Proceedings of the 10th IEEE Joint International Conference on Requirements Engineering (RE), 2002.

[13] N.E. Fenton, S.L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS, 1998.

[14] N. Maiden, C. Ncube. "Acquiring Requirements for COTS Selection", IEEE Software 15(2), 1998.

[15] X. Franch. "Systematic Formulation of Non-Functional Characteristics of Software". Procs. Proceedings of the 3rd IEEE International Conference on Requirements Engineering (ICRE), 1998.

[16] P. Botella, X. Burgués, X. Franch, M. Huerta, G. Salazar. "Modelling Non-Functional Requirements". Proceedings of Jornadas Ingeniería de Requisitos Aplicados (JIRA), 2001.

[17] X. Burgués, C. Estay, X. Franch, J. Pastor, C. Quer. "Combined Selection of COTS Components". Proceedings of the 1st International Conference on COTS-Based Software Systems (ICCBSS), LNCS 2255, 2002.

[18] G. Salazar-Zárate, P. Botella. "Use of UML for Non-Functional Aspects". Proceedings of 13th International Conference Software & Systems Engineering and their Applications (ICSSEA), 2000.

[19] G. Salazar-Zárate, P. Botella, A. Dahanajake. "Introducing Non-Functional Requirements in UML". In the book "*UML and the Unified Process*", Favre L. (editor), IRM Press, Hershey, PA, USA, to be published in the Spring of 2003.

[20] L.M. Cysneiros, J.C. Leite "Using UML to Reflect Non-Functional Requirements". Tutorial in the 10th IEEE Joint International Conference on Requirements Engineering (RE), 2002.