

COSTUME: A Method for Building Quality Models for  
Composite COTS-based Software Systems  
Carvalho, J.P. and Franch, X. and Grau, G. and Quer, C.  
Research Report LSI-04-12-R

Departament de Llenguatges i Sistemes Informàtics



UNIVERSITAT POLITÈCNICA DE CATALUNYA

# COSTUME: A Method for Building Quality Models for Composite COTS-based Software Systems

Juan P. Carvallo, Xavier Franch, Gemma Grau, Carme Quer  
Universitat Politècnica de Catalunya (UPC)  
c/ Jordi Girona 1-3 (Campus Nord, C6) E-08034 Barcelona (Catalunya, Spain)  
{carvallo, franch, ggrau, cquer}@lsi.upc.es

## Abstract

The use of quality models during the selection of Commercial, Off-The-Shelf (COTS) products provides a framework for the description of the domains which the COTS products belong to. Descriptions of COTS products and user quality requirements may be translated into the quality concepts defined in the model, making selection more efficient and reliable. In this paper we address the construction of quality models for Composite COTS-based Software Systems (CCSS), defined as systems that are composed by several interconnected COTS products. Selection processes carried out when procuring a CCSS require not a single COTS product to be selected but a set of them. As a consequence, instead of a classical quality model, we need a more elaborated one, defined as the composition of those models that belong to the domains of the COTS products that form the CCSS. We propose a method aimed at supporting the construction of quality models for CCSS as a composition of four activities. Most of the new quality features that appear in the CCSS model are defined in terms of the quality features of the component COTS products. As a result, we obtain quality models easier to understand, maintain and reuse, as well as compliant with the ISO/IEC 9126-1 quality standard.

## 1. Introduction

The amount of *Commercial, Off-The-Shelf (COTS) software products* [CL00] available in the market is growing more and more. This tendency is due to both the increasing adoption of component-based software technologies, and the continuous creation of new communication and marketing channels that bridge the gap among providers and consumers of those products. Therefore, there is an increasing need for identifying and qualifying the types of available COTS (hereafter, we abbreviate “COTS software product” by COTS) to improve the efficiency and reliability of their procurement [FSR96]. In particular, *quality requirements* have been recognized as crucial by the methods and processes proposed so far for driving this activity [Kon96, MN98]. Therefore, efforts are required to obtain, in an efficient way, reliable and comprehensive descriptions of COTS quality.

Quality models [ISO86] are a specially appealing way of structuring these descriptions. A *quality model* provides

a hierarchy of software *quality features* and *metrics* for computing their value. Quality models may be used in many contexts. For instance, [Dro96, KP96] use them for assessing the quality of custom systems.

In this paper we are interested in the use of quality models for supporting COTS selection. In this context, quality models are bound to *COTS domains*. Quality models shall embrace therefore those quality features that are common to all the COTS in that domain. Once the quality model is available, both the evaluation of COTS quality features and the quality requirements to be used during a selection process may be translated into concepts of the model (see fig. 1).

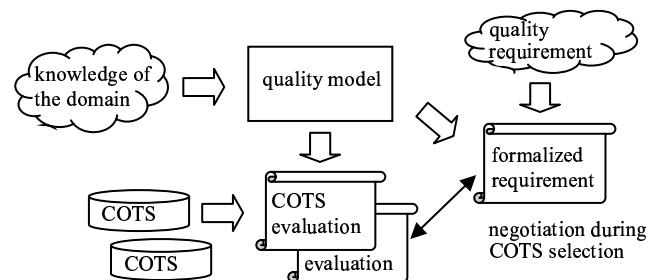


Figure 1 Using a quality model in COTS selection.

In previous work [FC02, FC03], we reported a method aimed at building quality models for COTS domains based on the ISO/IEC 9126-1 quality standard [ISO01], and we explored its usage in the context of COTS selection. In this paper, we extend considerably the scope of our previous work by addressing the construction of quality models for *composite COTS-based software systems* (hereafter, *CCSS*), i.e. systems that are composed of several COTS usually interconnected by some glue code. Examples of types of CCSS which our proposal can be applied to are *product lines* [CN02] and *cooperative information systems* [PS98]. Many CCSS fall into categories such as communication (mailing, video-conferences, etc.), document management and others that currently play an important role in the daily functioning of private companies and public organizations.

The objective of this paper is to identify and define the parts that compose quality models for CCSS (i.e., *CCSS quality models*) and to present the COSTUME method for building them.

## 2. Antecedents and Overview of COSTUME

The starting point of this research is a method for building quality models reported in [FC02, FC03]. We defined this method from both academic and industrial experiences in the field of COTS selection. Remarkably, in the case of industrial cases, we applied a lightweight approach of action-research [ALMN99] called *participatory observation*, in domains such as health-care systems and mail servers.

In [CFQ03] we reported a quality model for the mail server systems domain, which is the most complete case we have developed, encompassing dozens of subcharacteristics and more than 300 attributes. We were able to observe two particular experiences of mail server systems selection, in a public institution giving service to more than 30.000 people and in a small software consultant and ISP provider company. We expressed the requirements using the quality models, in some cases discovering ambiguities and incompleteness. We successfully used the model to describe the quality features of some products available in the market. The experience was positive, but we discovered some flaws in the models that resulted from our method, mainly:

- The obtained quality model considered mail server systems as isolated COTS. We realized that, due to the complexity of the domain, it is utterly important to identify the boundaries of the system more accurately, determining which people, organizations, and other software systems interact with it.
- In addition to their main functionality, i.e. supporting mailing facilities, those systems are supposed nowadays to offer other additional functions, such as videoconference infrastructure, virus detection, etc. We dealt with these features directly in the obtained mail server quality model itself, using the ISO/IEC *Suitability* and *Interoperability* subcharacteristics, increasing substantially the size of the model, damaging its understanding and endangering its reuse. We realized that we needed to improve our method by considering explicitly the case of quality models for CCSS. In fact, most COTS-based software systems are CCSS. There are several reasons for that:
  - Usually, system core requirements are very diverse in their nature and they embrace distinct functionalities, which are not covered by a single type of COTS.
  - General-purpose COTS such as anti-virus and compression tools, directory services, tracking tools, backup facilities, etc., have become widely established and their presence is assumed in most systems.
  - Successful COTS tend to incorporate through the years functionalities that were not originally related to them. Sometimes, even whole applications are incorporated to the upcoming versions. A usual reason

for this is that product suppliers often include features to differentiate their products from their competitors’.

As a result, we have defined COSTUME, a method aimed at defining CCSS quality models. COSTUME (*COMposite SofTware system qUality Model dEvelopment*) consists of four activities:

- *Activity 1. Analysing the environment of the CCSS.* The organizational elements that surround the CCSS are identified, as well as other external software systems which the CCSS interacts with. Relationships among the CCSS and the environment are established.
- *Activity 2. Decomposing the CCSS into subsystems.* The CCSS is decomposed into individual elements, each offering well-defined services. Services are identified with the help of the results obtained from the previous activity.
- *Activity 3. Building individual quality models for the CCSS.* We apply our former method [FC02, FC03] to build an ISO/IEC 9126-compliant quality model for each element of the CCSS.
- *Activity 4. Combining the individual quality models.* We obtain an ISO/IEC 9126-compliant quality model for the whole CCSS by the combination of the individual ones, obtaining therefore a single and uniform vision of the CCSS quality.

In the next sections, we will define more formally the activities and use a case study to present them in detail.

## 3. Analysing the Environment

CCSS do not operate in isolation; they communicate with other systems and with people, which act together as their *environment*. The first activity in COSTUME identifies the actors in the environment that interact with the CCSS.

We use in the rest of the paper a CCSS for mail server systems as example. *Mail server systems* (also known as *message servers*) are the core of the communication and coordination infrastructure of companies of any type. CCSS organized around mail servers systems are a good case study for many reasons, to name some: wide range of functionalities; strong links with their environment; intensive use world-wide; existence of an overwhelming number of mail-related products. Successful mail service deployment depends on its correct selection and a quality model may be used as the cornerstone for this process.

Actor	Abb.	Type	Goal
Mail Server System	MSS	Software	Provide communication infrastructure
Mail Client System	MCS	Software	Provide access to messages
Mail Server User	MSU	Human	Send and get messages
Mail Server Administrator	MSA	Human	Put mail server to work accurately and efficiently
Firewall	Fwll	Hardware	Filter incoming requests

**Table 1** Environmental actors of the mail server.

Table 1 presents the proposal of environmental actors for the mail server systems domain. We identify the actor's type and also give a short description of their goal.

The relationships among these actors can be depicted graphically using actor-based models. In our experiences, we have applied Yu's [Yu97] *i\** approach to actor-based modelling, and in particular its Strategic Dependency (SD) models for modelling networks of dependency relationships among actors in a system. Opportunities available to actors are explored by matching a *dependor*, which is the actor that "wants", and a *dependee* which has the "ability" to give what the dependor wants. Since the dependee's abilities can match the dependor's requests, a CCSS high-level dependency model can be developed. The *i\** approach was developed for modelling socio-technical systems that involve different software, human and organisational actors; therefore, it is specially well-suited for our needs.

Fig. 2 shows the *environmental model* for the mail server system case. Goals and resources have to be with the main functionalities that mail servers offer, namely mailing, cooperation facilities, address lists management and administrative duties. Soft goals are identified following the ISO/IEC characteristics and subcharacteristics (see 5.1); for instance, we find soft goals concerning security, efficiency and usability. As we are just interested in the part of the environment that involves the CCSS under analysis, dependencies among the actors of its environment are not of interest for our quality models. We remark that also the mail server behaviour may depend on environmental actors, as reflected by the two dependencies stemming from MSS to MSA.

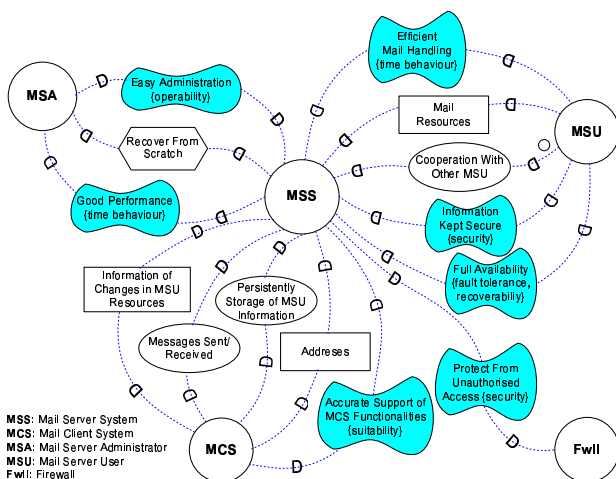


Figure 2 Environmental model for mail servers.

#### 4. Decomposing into Subsystems

A fundamental issue to tackle in COSTUME is to decompose properly the CCSS into individual components

to facilitate their separate analysis. However, we do not take a component-based approach but again an actor-based one: we identify the actors that play a role in the CCSS and the relationships among them. As a result, we break the system into its essentials, focusing on those services that actors provide instead of the physical arrangement of the system into components.

This activity is driven concurrently following two different carriers:

- Market-driven. Up-to-date knowledge of the types of tools currently available in the market and the functionalities they provide [BTV02]. Continuous analysis of white reports and technical documents from professional consultant companies is a key success factor.
- Goal-driven. Ability in identifying which are the actors in the system and their goals. Each actor has assigned a main goal to attain. Techniques for discovering goals [Ant96] can be tailored to our specific needs.

In our COSTUME, the dependencies in the environmental model drive the identification of some categories of actors. For instance, the goal dependency *Cooperation With Other MSU* points out the need of *groupware* oriented actors, and our knowledge of the market reveals the existence of meeting scheduling and *voice and video-conference* actors. In table 2 we show the actors identified, grouped by category (we omit actors' goals for space limitations). The third column shows the environmental model dependencies (already shown in fig. 2) that justifies the need of the actor.

Category	Actors	Rationale
Communication Support	Mail Servers	• Messages Sent/Received
	Routing Tools	• Efficient Mail Handling
Groupware Support	Meeting Scheduler Tools	• Co-operation With Other MSU
	Voice and Video-Conference Tools	
	Chatting Tools	
	Instant Messaging Tools	
	News Servers	
	Lists Servers	
Resources	Directory Services	• Mail Resources • Addresses • Persistent Storage of MSU information
	Compression Tools	• Mail Resources
Security Support	Anti-Spam Filter Managers	• Information Kept Secure
	Anti-virus Tools	
Administrative Support	Backup and Recovery Tools	• Full Availability
	Message Tracking Tools	• Efficient Mail Handling
	Configuration and Administration Tools	• Good Performance • Easy administration

Table 2 Actors for the mail servers case.

## 5. Building the Individual Quality Models

In [FC02, FC03] we proposed a method for building a ISO/IEC-compliant quality model for a COTS domain considered in isolation. This method follows some steps for tailoring a departing quality model proposed as part of the ISO/IEC 9126-1 standard [ISO01].

### 5.1 The ISO/IEC 9126-1 Quality Standard

The ISO/IEC 9126-standard has as main idea the use of quality models, composed of three types of quality features (characteristics, subcharacteristics and attributes), as a framework for software evaluation. The standard fixes a set of six characteristics (functionality, reliability, usability, efficiency, maintainability and portability) decomposed into a first level of such characteristics (such as security, portability, etc).

The standard is not precise at some points, as for example if multilevel hierarchies of subcharacteristics or attributes are allowed. For this reason we have felt compelled to take some decisions about which should be the organization of an ISO/IEC-based quality model. Some that are relevant for this paper are:

- Characteristics are non-measurable quality entities used to classify the rest of entities of the model.
- Subcharacteristics are quality entities that may be decomposed into other subcharacteristics or alternatively into attributes. Subcharacteristics are also used as a classification means.
- Hierarchies of subcharacteristics and attributes are allowed with no restrictions about number of levels.
- Attributes can be derived or basic. Basic attributes are directly measurable quality entities which can be objectively measured.
- Derived attributes are non-directly measurable quality entities. Sometimes it is not possible to find an objective metric to derive its value in terms of the basic attributes with which it is related. In these cases a subjective metric is required.

These decisions have been reflected in a conceptual model to represent a quality framework (as done for instance in [KHL01]), not included here for space limitations (see [FC02] for an earlier, similar version).

**Definition 1.** ISO/IEC 9126-1-compliant quality model.

An *ISO/IEC 9126-1-compliant quality model*  $QM$  is any valid instance of the software quality conceptual model which follows the decisions enumerated above.

The construction of individual quality models is exposed to multiple risk factors. In the following we explain some measures that may be applied to the usual found difficulties. As a first measure, we propose the use of ontologies for software quality [FGD02]; its use helps not only the construction of individual quality models but also its future composition in the next activity of

COSTUME. As a second measure, we propose the exhaustive reuse of quality features; although the situations in which the quality models are constructed are diverse, the quality models constructed for past experiences may be reused. The rest of models will have to be constructed for the particular selection process. We will come back to these questions in section 7.

### 5.2 Building Individual Quality Models

Starting from the six characteristics and their decomposition into subcharacteristics, our method for building individual quality models, as defined in [FC02, FC03]:

- Adds new subcharacteristics specific to the domain, refines the definition of some existing ones, or even eliminates some.
- Creates a hierarchy of quality subcharacteristics.
- Decomposes subcharacteristics into attributes, which keep track of observable features in the domain.
- Decomposes complex attributes into simpler ones, which are directly measurable.
- Determines metrics for the attributes.

**Definition 2.** Individual quality model.

An *individual quality model* is an ISO/IEC 9126-compliant quality model  $QM_D$  for a software domain  $D$ .

Construction of these individual quality models is endangered by many risk factors. We proposed several tips to help to overcome these usual difficulties. Among them, we mention the use of ontologies for quality models to provide a common framework beyond the standard [FGD02]; this use facilitates not only individual model construction but also their later combination in the next activity of COSTUME. The software quality ontology would include all the ISO/IEC 9126-related concepts represented in our software quality conceptual model.

On the other hand, the situation with respect to the construction of individual quality models is diverse. Those quality models already built from past experiences (i.e., involved in other types of CCSS) may be reused. The rest of the models have to be constructed and may be left incomplete and not refined until a particular procurement process requires more detail. We come back to these issues at section 7.

## 6. Composing the Individual Quality Models

Individual quality models give an exhaustive but individual, isolated and therefore incomplete view of parts of the CCSS. The next activity consists on composing these quality models to provide an integrated view of the quality of the whole CCSS. Composition means combining appropriately the quality features of the individual quality models. More formally:

**Definition 3.** CCSS quality model.

Let  $S$  be a CCSS and let  $A$  be the set of actors which  $S$  is decomposed into. A *CCSS quality model* for  $S$  is defined as a tuple  $QM_{ccss} = (\{QM_D\}_{D \in A}, QM_S, Map_S)$  such that:

- $\{QM_D\}_{D \in A}$  are individual quality models considering the actors of  $A$  as domains. We call them *component quality models*.
- $QM_S$  is an ISO/IEC 9126-compliant quality model. We call it *compound quality model*.
- $Map_S$  is a family of 2 mappings,  $Map_S = (MapSubcars_S, MapAttrs_S)$ , mapping some of the quality subcharacteristics resp. attributes from  $QM_S$  to those in  $\{QM_D\}$ .

Characteristics are not explicitly taken into account because we consider that the compound quality model includes the six ones defined in the ISO/IEC 9126-1.

The crucial fact is that the elements of the composite quality model construction are mostly defined as the result of a mapping from the new quality elements to the existing ones, and just a few elements emerge. In its turn, this mapping is built from the repeated application of some subcharacteristic and attribute combination patterns defined in the rest of the section.

**6.1 Combination patterns for subcharacteristics**

The subcharacteristics are high level quality features and they should be defined in the ontology proposed in the previous section. Our approach is not restricted to any specific ontology, the ontology may even not exist, but we propose its existence as a useful conceptual tool that facilitates the construction of quality models for CCSS. Specifically, if the individual quality models are constructed using the ontology, the application of combination patterns become an easier task.

Let  $QM_S$  be the compound quality model and let  $QM_1, QM_2 \in \{QM_D\}$  be two component quality models of a given CCSS  $S$ . We identify 4 patterns for combining subcharacteristics presented below (see figure 3).

Pattern	$QM_1$	$QM_2$	$QM_S$
Identification			
Nesting			
Abstraction		N/A	
System	N/A	N/A	

Figure 3 Subcharacteristic patterns

For simplicity, definitions are given on pairs of quality elements; generalization to  $n$  elements is straightforward. Prefixing is used when necessary, represented by “:”. We use the parent function to obtain the parent of a given quality feature in the corresponding hierarchy. We do not bind the definition to the existence of any software quality ontology, although its existence will allow easier pattern identification and application. The definitions do not handle explicitly the case where subcharacteristics parents are characteristics; extension is straightforward.

Combination patterns are supposed to be applied top-down, meaning that first they are applied to build the upper levels of the resulting hierarchy and then the lower levels are fulfilled stepwise; of course, during quality model construction, some degree of intertwining and iteration arises, and the condition is relaxed to: a quality entity may appear in the compound model only if its parent has already been inserted therein.

**Identification pattern**

- Precondition:
  - 1)  $x$  and  $y$  are two subcharacteristics,  $x \in QM_1, y \in QM_2, QM_1 \neq QM_2$ .
  - 2) exists a subcharacteristic  $p \in QM_S$  such that  $MapSubcars_S(p) = \{\text{parent}(x), \text{parent}(y)\}$ .
  - 3)  $z \notin QM_S$ .
- Pattern application: Identification( $x, y, z, p$ )
- Postcondition:
  - 1)  $z \in QM_S$  such that  $\text{parent}(z) = p$ .
  - 2)  $MapSubcars_S(z) = \{QM_1::x, QM_2::y\}$ .

**Rationale**

There are many subcharacteristics that appear repeatedly in a great deal of quality models. In this case, we simply add a subcharacteristic in the compound model. As will be the general case, the pattern definition allows different names for the subcharacteristic in different models, although the usual case in this pattern will be having the same name, especially if an ontology has been used.

**Example**

A basic case is the ISO/IEC 9126-1 subcharacteristics themselves, e.g. *Security*. More specifically to our case study, both the quality models of *Voice and Video-Conference* and *Chatting* domains decompose their *Security* subcharacteristic into *Local Security* and *External Security*. We have applied twice the pattern to obtain the same subcharacteristics in the compound model.

**Nesting pattern**

- Precondition:
  - 1)  $x$  and  $y$  are two subcharacteristics,  $x \in QM_1, y \in QM_2$ . In this case,  $QM_1$  and  $QM_2$  may be the same.
  - 2) exists a subcharacteristic  $p \in QM_S$  such that  $MapSubcars_S(p) = \{\text{parent}(x), \text{parent}(y)\}$ .

3)  $x, y, z \notin QM_S$ ,

- Pattern application: Nesting( $x, y, z, p$ )
- Postcondition:

1)  $x, y, z \in QM_S$  such that  $\text{parent}(z) = p$ ,  $\text{parent}(x) = \text{parent}(y) = z$ .

2)  $\text{MapSubcars}_S(x) = \{QM_1::x\}$ ,  $\text{MapSubcars}_S(y) = \{QM_2::y\}$ .

3)  $z \notin \text{dom}(\text{MapSubcars}_S)$

#### Rationale

Many other subcharacteristics, on the contrary, are to be kept separately in the compound model because they keep track of distinct quality features. However, it is not enough with adding them because the resulting model would be too flat. We identify a new subcharacteristic ( $z$  in the definition above) that aids on structuring the compound model. The new subcharacteristic may show up following two different criteria:

- Domain-oriented. The new subcharacteristic  $z$  puts together two subcharacteristics  $x$  and  $y$  of the same domain (i.e.,  $QM_1$  and  $QM_2$  are the same).
- Feature-oriented. The new subcharacteristic  $z$  puts together two subcharacteristics  $x$  and  $y$  of two different domains (i.e.,  $QM_1$  and  $QM_2$  are different) that address to a similar feature. In this case, the new subcharacteristic represents a new concept and the ontology should be updated.

#### Example

A domain-oriented application of the pattern appears when considering the *Accuracy* subcharacteristics *Accurate Scanning&Repair* and *Actualization of Lists* from the anti-virus domain. They have been grouped in the compound model by introducing a new subcharacteristic *Antivirus Accuracy*, defined as child of *Accuracy* in this model.

A feature-oriented case appears when considering the *Security* subcharacteristic *User Privileges* from directory servers and *Password Management* from backup and recovery domains. We have combined them in the compound model by defining a parent subcharacteristic called *Internal Security*, to be added to the ontology if exists.

#### Abstraction pattern

- Precondition:

1)  $x$  is a subcharacteristic,  $x \in QM_1$ .

2) exists a subcharacteristic  $p \in QM_S$  such that  $\text{MapSubcars}_S(p) = \{\text{parent}(x)\}$ .

3)  $x, z \in QM_S$ ,

- Pattern application: Abstraction( $x, z, p$ )

- Postcondition:

1)  $x, z \in QM_S$  such that  $\text{parent}(z) = p$ ,  $\text{parent}(x) = z$ .

2)  $\text{MapSubcars}_S(x) = \{QM_1::x\}$ .

3)  $z \notin \text{dom}(\text{MapSubcars}_S)$

#### Rationale

This is a special case of the previous one. It appears mainly in small domains in which a high-level subcharacteristic is decomposed directly into attributes. When incorporating this subcharacteristic into the compound quality model, which may be already decomposed into other subcharacteristics, we consider convenient to introduce a new one grouping those attributes for leveraging the model.

#### Example

The *Fault Tolerance* subcharacteristic is not further decomposed in the domain of routing tools; attributes such as *Time to Recalculate Path* are directly children of this subcharacteristic. However, when building the compound model, we considered convenient to define a new subcharacteristic *Adaptable Routing* as child of *Fault Tolerance* enclosing the attributes mentioned above.

#### System pattern

- Precondition:

1)  $x$  is a subcharacteristic,  $x \in QM_S$ .

- Pattern application: System( $x$ )

- Postcondition:

1) “System  $x$ ”  $\in QM_S$  such that  $\text{parent}(\text{“System } x\text{”}) = x$ .

2) “System  $x$ ”  $\notin \text{dom}(\text{MapSubcars}_S)$

#### Rationale

This pattern allows to introduce a subcharacteristic for grouping those attributes whose definition varies substantially from the ones in the individual quality models, or even new attributes (e.g., coming from system architecture properties).

#### Example

In the compound model usually will appear a new subcharacteristic called *System Interoperability*, which will group attributes to evaluate the capability of products of the compound domains of interoperate among them.

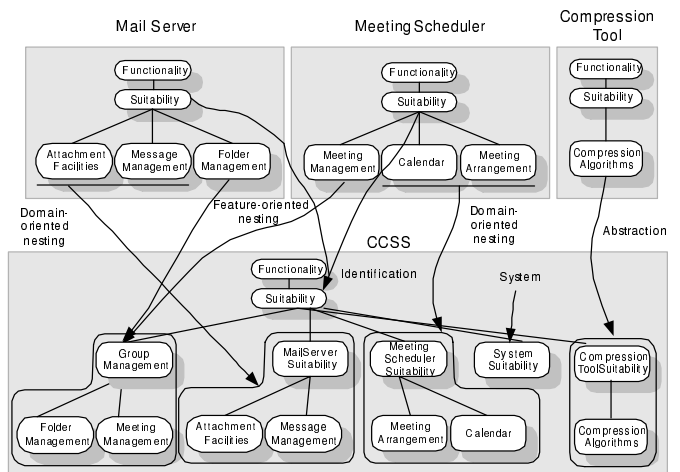


Figure 4 Example of application of subcharacteristic patterns

Figure 4 shows a final example in the part of suitability that makes use of the 4 patterns. The identification pattern is used to identify the ISO/IEC *Suitability* subcharacteristic of all the individual domains. The domain-oriented style of the nesting pattern is used to keep track of that part of suitability inherent of mail servers (i.e., *Message Management*) and meeting schedulers (i.e., *Meeting Arrangement and Calendar*). The feature-oriented style of this pattern is applied to put together related subcharacteristics such as Folder and Meeting Management that can be both viewed as management of group of items (messages and meetings, respectively). The abstraction pattern introduces a new subcharacteristic to put the suitability of compression algorithms at the same level as the others. Finally, the system pattern is used to provide room for prospective attributes.

We remark that, in earlier versions of COSTUME, we had studied other subcharacteristic patterns, which have finally been discarded: although they were theoretically feasible they've not become necessary in our case studies.

The patterns defined in this section may be applied in different ways. We have found that a strategy that behaves well in most cases can be defined as follows:

- The identification pattern is used in the upper levels of the quality model hierarchy.
- The domain-oriented nesting pattern is applied in the lower levels of the subcharacteristic hierarchy. Occasionally, some identification patterns may be used also there, when combining attributes from similar domains (e.g., anti-virus and anti-spam).
- The abstraction pattern is used to leverage the hierarchy.
- The system pattern is applied only to the first-level subcharacteristics, to avoid proliferation.
- The feature-oriented nesting pattern is used only when a new concept has been identified in the compound model.

With this strategy, pattern application is a mechanical process except for the last case, whose rate is below 15% in our experiences, allowing tool support as in the related field of ontology merging.

## 6.2 Combination patterns for attributes

CCSS quality attributes of the compound model are basically of three different sorts. Firstly, some attributes are simply inherited as they are from the component models. Secondly, some attributes represent ontological concepts that were already present in some component models but they need to be redefined. Lastly, some attributes represent new quality features that need to be added in the compound quality model. The attribute patterns catch these three situations.

Let  $QM_S$  be the compound quality model and let  $QM_1, QM_2 \in \{QM_D\}$  be two component quality models of a given CCSS  $S$ . We identify 5 patterns for combining attributes. As with subcharacteristics, we define them over pair of attributes, to be extended to the general case. Also to mention, preconditions are assuming the case that the involved attributes are children of subcharacteristics; more elaborated ones for taking into account also the case of being children of attributes are straightforward.

### Delegation pattern

- Precondition:
  - 1)  $x$  and  $y$  are two attributes,  $x \in QM_1, y \in QM_2, QM_1 \neq QM_2$ ,
  - 2) exists a subcharacteristic  $p \in QM_S$  such that  $MapSubcars_S(p) = \{\text{parent}(x), \text{parent}(y)\}$ .
  - 3)  $z \notin QM_S$ .
- Pattern application:  $Delegation(x, y, z, p)$
- Postcondition:
  - 1)  $z \in QM_S$  such that  $\text{parent}(z) = p$ .
  - 2)  $MapAttrs_S(z) = \{QM_1::x, QM_2::y\}$ .
  - 3)  $z$  is defined as  $z = QM_1::x$ .

### Rationale

Concerning a particular quality attribute, it may be the case that one of the component COTS prevails over the rest. Therefore, the attribute in the compound model is defined just in terms of this prevalent one.

### Example

An example of delegation comes with the *Existence of Log File* (EoLF) attribute. Although every component in the CCSS may generate log files, we are particularly interested in knowing if the mail server component exhibits this feature and the definition is:

$$CCSS::EoLF = MailServer::EoLF$$

### Redefinition pattern

- Precondition:
  - 1)  $x$  and  $y$  are two attributes,  $x \in QM_1, y \in QM_2, QM_1 \neq QM_2$ ,
  - 2) exists a subcharacteristic  $p \in QM_S$  such that  $MapSubcars_S(p) = \{\text{parent}(x), \text{parent}(y)\}$ .
  - 3)  $z \notin QM_S$ .
- Pattern application:  $Redefinition(x, y, z, p)$
- Postcondition:
  - 1)  $z \in QM_S$  such that  $\text{parent}(z) = p$ .
  - 2)  $MapAttrs_S(z) = \{QM_1::x, QM_2::y\}$ .
  - 3)  $z$  is defined as  $z = K$ .

### Rationale

This pattern is applied when a quality concept appearing in (some of) the component models is valid in the compound model, but the definition in the compound model does not depend on the component models' values. Instead, the new attribute is a basic one.

### Example

*Time in the Market* is an attribute belonging to the *Maturity* subcharacteristic. Each component model will include this attribute, and so the compound one will too, but the value of this last attribute is obviously not computed from the others.

### Combination pattern

- Precondition:
  - 1)  $x$  and  $y$  are two attributes,  $x \in QM_1, y \in QM_2$ .
  - 2) exists a subcharacteristic  $p \in QM_S$  such that  $MapSubcars_S(p) = \{parent(x), parent(y)\}$ .
  - 3)  $z \notin QM_S$ .
- Pattern application:  $Combination(x, y, z, p)$
- Postcondition:
  - 1)  $z \in QM_S$  such that  $parent(z) = p$ .
  - 2)  $MapAttrs_S(z) = \{QM_1::x, QM_2::y\}$ .
  - 3)  $z$  is defined as  $z = f(QM_1::x, QM_2::y)$ .

### Rationale

The most usual case in considering a quality concept consists on taking all the component quality attributes into account and to define a metrics for combining them. This situation reflects our observation (and the intuition) that system quality attributes highly depend of the attributes of its components.

### Examples

Many examples of attribute combination exist, with different combination functions  $f$ .

- Some combination functions are additive ones, remarkably the sum. For instance, the *Total Average Time to Send a Message* (attribute belonging to the *Time Behaviour* subcharacteristic), is defined as the sum of five attributes. They are *Average Sending Time* from mail servers domain, *Virus Scanning Time* from anti-virus, *Mail Compression Time* for data compression tools, *Time to Find Destination Node* from routing tools domain, and *Time to Update Log File* from mail servers again.
- When putting together quality models, sometimes we are interested as combination functions in the maximum or minimum values of some attribute. For instance, this is the case of the *Mean Time Between Failures* attribute, defined as the minimum of the mean time between failures of the components.
- Some quality attribute have sets as values, and the corresponding attribute in the compound model may be defined as the union or intersection of such sets. One example shows up in the *Languages Of Documentation* attribute from the *Understandability* subcharacteristic, defined as the intersection of the component models' attributes.
- In fact, composition may be more complex in other cases. For instance, we have an interoperability attribute named *Communication Protocols* (CP) related to the way that component COTS interoperate with each other. The

attribute defined in the component domains as a function over one variable (for a domain  $x$ ,  $x::CP(y)$  states the communication protocols from  $x$  to the domain  $y$ ), which turns out to be a function with two variables in the compound model, with the following definition:

$$CCSS::CP(x, y) = x::CP(y).$$

### New combined attribute pattern

- Precondition:
  - 1)  $x$  and  $y$  are two attributes,  $x \in QM_1, y \in QM_2$ .
  - 2) exists a subcharacteristic  $p \in QM_S$  such that  $MapSubcars_S(p) = \{parent(x), parent(y)\}$ .
  - 3)  $z \notin QM_S$ .
- Pattern application:  $NewCombinedAttribute(x, y, z, p)$
- Postcondition:
  - 1)  $z \in QM_S$  such that  $parent(z) = p$ .
  - 2)  $z \notin \text{dom}(MapAttrs_S)$ .
  - 3)  $z$  is defined as  $z = f(QM_1::x, QM_2::y)$ .

### Rationale

Some times new derived attributes are needed in the compound model, closely related with other existing ones. In this case, although the new attribute is not conceptually equivalent to others, its metrics will keep track of this dependency. Ontology update may be necessary if it is considered that the new attribute represents an important software quality concept.

### Example

An attribute belonging to *Portability* is *Operating Systems Supported*, defined in each component quality models as a set of labels. The most obvious way to define this attribute in the compound model is as the intersection of such sets, but we find this definition too restrictive, since an implementation of the mail server CCSS may deploy its components in different nodes with different operating systems (in fact, this is the usual case). For this reason, we define a new attribute *Feasible Operating Systems Combinations* which registers every admissible combination of components into nodes and of operating systems for these nodes. This attribute, defined as a set of sets of labels, is utterly important in the assessment of candidate deployment platforms.

### New system attribute pattern

- Precondition:
  - 1) exists a subcharacteristic  $p \in QM_S$ .
  - 2)  $z \notin QM_S$ .
- Pattern application:  $NewSystemAttribute(z, p)$
- Postcondition:
  - 1)  $z \in QM_S$  such that  $parent(z) = p$ .
  - 2)  $z \notin \text{dom}(MapAttrs_S)$ .
  - 3)  $z$  is defined as  $z = K$ .

### Rationale

This last case appears when an attribute is not related at all with quality features of the component models. Most of

the times the pattern will be applied to capture architectural properties of the CCSS. It should be mentioned that according to subcharacteristic patterns, most of the cases the  $p$  subcharacteristic mentioned in the definition will be one coming from a system subcharacteristic pattern, but we prefer not to force that.

#### *Exemple*

Some classical object-oriented measures, such as cohesion, are introduced in the *Maintainability* subcharacteristic of the system as attributes. *Cohesion* does not depend on the quality features of the model, but instead it has to be with the way the COTS are interconnected in the CCSS.

## 7. Analysis of COSTUME

At a first glance, the construction of so many individual quality models may seem time-consuming and cumbersome but in fact, in this section we argue that it pays off. On the one hand, it supports return on investment through model reuse. On the other hand, their construction can be requirements-driven and therefore, individual quality models may be left incomplete. Last, the internal structure of the CCSS quality model is well-suited for the unavoidable maintenance of the model, coming from the continuous changes on the COTS market

### 7.1 Requirements-Driven Construction

Due to the usual time pressures in real-life project, it seems illusory to ask for the complete development of individual quality models. Instead, we propose a *requirements-driven* construction strategy that concentrates the effort on the quality factors directly related with the specific requirements of the selection project which the model is being constructed for. Therefore, we consider an iterative selection process [MN98, BEFPQ02] in which requirements and evaluation through the construction of the quality model are highly intertwined.

Not only the type of requirements but also the level of detail must be considered. This second guideline aligns with the observation that requirements on COTS that do not provide the core functionalities of the CCSS are usually not totally detailed. For instance, requirements can be as vague as “The messages sent by the system shall be not infected by virus” without specifying e.g. which actions are required once the virus is detected. The quality model for the anti-virus domain may be reduced to a few attributes.

### 7.2 Reuse of Quality Models

However, the requirements-driven vision by itself is too narrow for our purposes. We do not think in quality models as throw-away artefacts, that is, models built just

for the project that originate them. On the contrary, we think that quality models as a whole, or parts of them, can be reused in a large number of projects. Specifically, we can apply reuse in the following situations:

- Compound quality models can be reused in different selection processes of the same type of CCSS. Since the requirements will probably be different, the model should be enlarged to embrace the quality features left to cover these requirements.
- Component quality models can be reused for building new CCSS quality models. This is specially true in the case of quality models for general-purpose domains, such as anti-virus, and backup and restore tools: once built, they can be reused in a great deal of CCSS quality models. In the extreme case, even for non-CCSS selection (i.e., selection of single COTS) we can reuse the individual quality model directly.
- Finally, some results of the study of the environment can be reused in the construction of other composite quality models where the same actors appear. For instance, in case we want to construct a composite quality model for a *Mail Client System*, dependencies in the environmental model respect to *Mail Server Systems* can be reused.

It remains implicit that we assume the existence of a repository of quality models, which can be reused as they are in selection processes, or used as a starting point for the construction of new quality models. Also, we assume that this repository could be updated with new quality models. We propose in [CFQT04] the organization of this repository as a taxonomy in which related domains may share the common parts of their quality models. In fact, taking reuse into account, we could redefine the activity 3 of COSTUME (building an individual quality model for a domain) encompassing the following steps:

**Step 1: Localization.** Look for an individual quality model for the domain in the repository. The organization of the repository as a taxonomy of domains supports this search. If not found, search for the closest domain in the taxonomy and take its quality model as starting point. If no domain is found, take the ISO/IEC 9126-1 as starting point.

**Step 2: Tailoring.** Enlarge this departing quality model taking into account the characteristics of the domain and the type of requirements of the project. This enlargement is done by applying the method for the construction of individual quality models of section 5.2 in a requirements-driven way. Therefore, just those quality entities related with the usual requirements of a CCSS will be added. On the other hand, the already-existing quality features that do not apply for the selection case will not take a part in the selection process.

**Step 3: Refactoring.** For enhancing future reuse, once the individual models are built, the taxonomy should be updated. The actions to be taken are many: to split nodes of the taxonomy, to create new ones, to leave out some quality features too tight to the ongoing project, or even to complete the quality model beyond the project reqts.

### 7.3 Maintenance of CCSS quality models

Constant change is a major characteristic around COTS market. Releases and versions of products appear in a few months, incorporating new functionalities, and improving existing ones. With each new version, new quality features come into existence, which shall be incorporated into the quality models.

The internal structure of quality models supports the identification and classification of new quality features. But in addition to this property, common to any quality-model-based approach, our CCSS quality models are specially well-suited for the addition of new groups of functionalities, through the definition of new actors and the actualisation of the mappings. If mail servers history is examined, it can be checked that former systems didn't provide some functionalities, such as meeting scheduling or anti-virus (not a main goal of mail servers). In the near future, one can envisage some groups of new functionalities coming up, such as voice recognition (for dictating mails) and domain-ontology alignment (e.g., for organizing folders or determining mail subjects). The new actors generate new individual quality models that do not interfere with the existing ones. The CCSS quality model is updated taking into account this new models.

## 8. Conclusions and Future Work

In this paper we have proposed a method to create quality models for composite COTS-based software systems (CCSS). The method, COSTUME, is organized into four activities that encompass: environment analysis; CCSS decomposition; construction of quality models for the components; combination of the quality models to obtain the one of interest. Quality features of the compound quality model are defined in terms of those in the individual quality models that are bound to the actors inside the CCSS.

We think that the contributions of our work are:

- We define a precise method for complex quality model construction. Some of them exist (not as many as one could think) [Dro96, KP96], including ours [FC02, FC03], but they are not specifically oriented for this kind of composite system we are addressing in the paper. Precision of the method strongly relies on pattern application. In the case of subcharacteristics, patterns can be applied almost in an automatic way. In the case of attributes, the

patterns provide a useful framework for classifying the new, arising attributes of the quality model.

- The process for building the quality model is efficient, in the sense that it does make intensive reuse. We have showed in section 7 three different ways of reuse, that make our approach less time-consuming than others. Also, our requirements-driven approach avoids spending time in those parts of the model that are not of interest for the current selection process.
- Market evolution requires changes in the quality models, but separation of concerns inside the compound quality model avoids unnecessary modification of existing quality models. On the other hand, CCSS quality models are highly traceable, which is one fundamental characteristic. In other words, the definition keeps track of which quality features in component COTS affect which quality features in the CCSS. Traceability is a property that supports maintainability.
- Our definition of CCSS quality model is a dual one, in the sense that it is highly structured inside, by keeping track of component models, but also is ISO/IEC 9126-1-compliant, which can make it more attractive for practitioners.

For these reasons, we argue that the construction of quality models for its use in COTS selection usually pays off. In fact, quality models are used either explicitly or implicitly in any trustable selection experience. Also, we would like to enumerate other contexts in addition to selection that may benefit of the existence of quality models: *system development* where quality attributes may be used to guide system development and quality assessment procedures; *product quality assessment and certification*; *market exploration*, where quality attributes can help providers to know which properties would be more interesting for buyers of their new product versions; and *reference model construction*, for those organisations who base their revenues in selling product reports and white papers. These other contexts of use make more critical the existence of methods as COSTUME for developing quality models.

We are currently developing tool support for our COSTUME. We have a first prototype for building individual quality models [QM03], whose extension to implement the whole method is a short-term goal, giving support especially to pattern application. As future work, we plan to integrate more tightly the ontology into the method.

## References

- [ALMN99] D. Avison, F. Lau, M. Myers, P.A. Nielsen. "Action Research". CACM 42(1), 1999.
- [Ant96] Annie I. Antón. "Goal-Based Requirements Analysis". In *Procs. 2<sup>nd</sup> IEEE ICRE*, 1996.

- [BEFPQ02] X. Burgués, C. Estay, X. Franch, J.A. Pastor, C. Quer. "Combined Selection of COTS Components". *Procs. 1<sup>st</sup> ICCBSS, LNCS 2255*, 2002.
- [BTV02] M. Bertoa, J.M. Troya, A. Vallecillo. "A Survey on the Quality Information Provided by Software Component Vendors". 7th ECOOP QAOOSE workshop, 2002.
- [CFQT04] J.P. Carvallo, X. Franch, C. Quer, M. Torchiano. "Characterization of a Taxonomy for Business Applications and the Relationships among them". In *Procs. 3<sup>rd</sup> ICCBSS, LNCS*, 2004 (to appear).
- [CFQ03] J.P. Carvallo, X. Franch, C. Quer. "Defining a Quality Model for Mail Servers". *Procs. 2<sup>nd</sup> ICCBSS, LNCS 2580*, 2003.
- [CL00] D. Carney, F. Long. "What Do You Mean by COTS? Finally a Useful Answer". *IEEE Software*, 17(2), March 2000.
- [CN02] P. Clements, L. Northrop. *Software Product Lines: Practices and Patterns*. SEI series in SE, Addison-Wesley 2002.
- [Dro96] R.G. Dromey. "Cornering the Chimera". *IEEE Software*, 13(1), 1996.
- [FC02] X. Franch, J.P. Carvallo. "A Quality-Model-Based Approach for Describing and Evaluating Software Packages". *Procs. 10<sup>th</sup> IEEE RE*, 2002.
- [FC03] X. Franch, J.P. Carvallo. "Using Quality Models in Software Package Selection". *IEEE Software*, 20(1), 2003.
- [FGD02] R.A. Falbo, G. Guizzardi, K.C. Duarte. "An Ontological Approach to Domain Engineering". *Procs. 14<sup>th</sup> SEKE*, 2002.
- [FSR96] A. Finkelstein, G. Spanoudakis, M. Ryan. "Software Package Requirements and Procurement". *Procs. 8<sup>th</sup> IEEE IWSSD*, 1996.
- [ISO86] *ISO Standard 8402: Quality management and quality assurance-Vocabulary*, 1986.
- [ISO01] *ISO/IEC Standard 9126-1 Software Engineering – Product Quality – Part 1: Quality Model*, 2001.
- [KHL01] B. Kitchenham, R. Hugues, S.G. Linkman. "Modeling Software Measurement Data". *IEEE TSE*, 27(9), 2001.
- [Kon96] J. Kontyo. "A Case Study in Applying a Systematic Method for COTS Selection". *Procs. 18<sup>th</sup> IEEE ICSE*, 1996.
- [KP96] B. Kitchenham, S.L. Pfleeger. "Software Quality: the Elusive Target". *IEEE Software*, 13(1), 1996.
- [MN98] N. Maiden, C. Ncube. "Acquiring Requirements for COTS Selection". *IEEE Software*, 15(2), 1998.
- [PS98] M. Papazoglou, G. Schlageter (eds.). *Cooperative Information Systems: Trends & Directions*. Academic Press, 1998.
- [QM03] QM Tool. "<http://www.lsi.upc.es/%7Egessi/QMTool/QMTool.html>"
- [Yu97] E. Yu. "Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering". *Procs. 3<sup>rd</sup> IEEE ISRE*, 1997.