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), 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 propose a method for 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.

1. Introduction

The amount of *Commercial, Off-The-Shelf (COTS) software products* [1] 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 [2]. In particular, *quality requirements* have been recognized as crucial by the methods and processes proposed so far for driving this activity [3, 4]. Thus, efforts are required to obtain, in an efficient way, reliable and comprehensive descriptions of COTS quality.

Quality models [5] are an especially appealing way of structuring these descriptions. A *quality model* provides a hierarchy of software *quality features* and *metrics* for computing their value. They are used in many contexts, e.g. for assessing the quality of custom systems [6, 7].

In this paper we present COSTUME (*CO*mposite *S*of*T*ware system q*U*ality *M*odel d*E*velopment), a method for 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. The quality model for the CCSS will be built upon the quality models bound to the domains of its component COTS. Examples of types of CCSS are *product lines* [8] and *cooperative information systems* [9]. In fact, most current 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, etc., have become widely established and their presence is assumed in most systems.
- Successful COTS tend to incorporate through the years services not originally related to them.

The COSTUME method 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 COTS domains*. The CCSS is decomposed into COTS domains, each offering well-defined services, which are identified with the help of the results coming from activity 1.
- *Activity 3. Building individual quality models for the COTS domains*. We apply our former IQMC method [10] to build an ISO/IEC 9126-compliant quality model for each COTS domain of the CCSS.
- *Activity 4. Composing 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.

Proceedings of the Fourth International Conference on Quality Software (QSIC'04)
0-7695-2207-6/04 $ 20.00 IEEE

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

2. 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 *mail server* CCSS as example. Mail server CCSS 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. Table 1 presents the proposal of environmental actors for the mail server CCSS that we use through the paper. We identify the actors' type and also give a short description of their main goal. The relationships among the mail server CCSS and the rest of actors can be depicted graphically using a table such as Table 2. We represent dependencies concerning functionalities (e.g., address lists management), non-functional properties (e.g., security), data (e.g., mail resources) and tasks (e.g., recovery).

Table 1. Mail server CCSS environmental actors.

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 2. Relationships in the mail server CCSS.

MSS depends on	Relationship
MSA	Recover from scratch
	Good performance
Fwll	Protect from unauthorised access
Others depend on MSS	**Relationship**
MSA	Easy administration
MSU	Efficient mail handling
	Mail resources
	Cooperation with other MSU
	Information kept secure
	Full availability

3. Decomposing the CCSS into COTS domains

A fundamental issue to tackle in COSTUME is to decompose properly the CCSS into COTS domains 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 (that represent COTS domains), 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 services that they provide. 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 [11] can be tailored to our specific needs.

In COSTUME, the dependencies in the environmental model drive the identification of some categories of actors. For instance, the 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 domains. In Table 3 we show the actors identified, grouped by category (we omit actors' goals for space limitations). The third column shows the environmental model relationships that justify the need of the actor.

Table 3. Actors for the mail server CCSS.

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 & Recovery Tools	• Full Availability
	Message Tracking Tools	• Efficient Mail Handling • Good Performance • Easy administration
	Configuration and Administration Tools	

4. Building the Individual Quality Models

In [10] 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 [12]. As a result we obtain ISO/IEC 9126-compliant quality models for these domains.

Definition 1. ISO/IEC 9126-compliant quality model.
An *ISO/IEC 9126-compliant quality model QM* is any extension of the ISO/IEC 9126-1 quality model that follows the concepts stated in this standard.

Definition 2. Individual quality model.
An *individual quality model* is an ISO/IEC 9126-compliant quality model QM_D for a COTS domain D.

The situation with respect to the construction of individual quality models is diverse. Those quality models already built from past experiences may be reused. The rest of the models must be constructed and may be left incomplete, to be refined when a particular procurement process requires more detail. See section 6 for details.

5. 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 $QM_S = (\{QM_D\}_{D \in A}, QM_S, Map_S)$ 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 $\{QM_D\}$[1].

The crucial fact is that the elements of the composite quality model 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 composition patterns defined in the rest of the section.

For simplicity, definitions are given on pairs of quality elements and pairs of compound quality models; generalization to n elements is straightforward but technically cumbersome. When the opposite is not indicated, the patterns can also be applied when the two quality models involved are the same. 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 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.

Composition 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.

5.1 Composition patterns for subcharacteristics

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 6 patterns presented below for combining subcharacteristics (see Table 4).

Table 4. Subcharacteristic patterns

Pattern	QM_1	QM_2	QM_S
Identification	p x	q y	r z
Combination	p x	q y	r u v
Nesting	p x	q y	r z u v
Drag-and-drop	p x	N/A	r u
Abstraction	p x	N/A	r z u
System	N/A	N/A	r z

[1] 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.

Identification pattern

Precondition:

1) p, q, x and y are four subcharacteristics, $\{p, x\} \subseteq QM_1$, $\{q, y\} \subseteq QM_2$, parent(x) = p, parent(y) = q, $QM_1 \neq QM_2$.

2) exists a subcharacteristic $r \in QM_S$ such that $\{p, q\} \subseteq MapSubcars_S(r)$.

3) $z \notin QM_S$.

Pattern application: Identification(p, q, x, y, r, z)

Postcondition:

1) $z \in QM_S$ such that parent(z) = r.

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.

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* tools 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.

Combination pattern

Precondition:

1) p, q, x and y are four subcharacteristics, $\{p, x\} \subseteq QM_1$, $\{q, y\} \subseteq QM_2$, parent(x) = p, parent(y) = q.

2) exists a subcharacteristic $r \in QM_S$ such that $\{p, q\} \subseteq MapSubcars_S(r)$.

3) u, $v \notin QM_S$.

Pattern application: Combination(p, q, x, y, r, u, v)

Postcondition:

1) u, $v \in QM_S$ such that parent(u) = parent(v) = r.

2) $MapSubcars_S(u) = \{QM_1::x\}$, $MapSubcars_S(v) = \{QM_2::y\}$.

Rationale. Many subcharacteristics are to be kept separately in the compound model because they keep track of distinct quality features.

Example. Consider the *Verification Capabilities subcharacteristics* that appear in the anti-spam and anti-virus domains. They include the subcharacteristics *Verification of Spam Detection* and *Verification of Virus Detection* respectively. These subcharacteristics have been kept separately in the compound model below a *Verification Capabilities* subcharacteristic, which has been obtained by a former application of the identification pattern.

Nesting pattern

Precondition:

1) p, q, x and y are four subcharacteristics, $\{p, x\} \subseteq QM_1$, $\{q, y\} \subseteq QM_2$, parent(x) = p, parent(y) = q.

2) exists a subcharacteristic $r \in QM_S$ such that $\{p, q\} \subseteq MapSubcars_S(r)$.

3) z, u, $v \notin QM_S$.

Pattern application: Nesting(p, q, x, y, r, z, u, v)

Postcondition:

1) z, u, $v \in QM_S$ such that parent(z) = r, parent(u) = parent(v) = z.

2) $MapSubcars_S(u) = \{QM_1::x\}$, $MapSubcars_S(v) = \{QM_2::y\}$.

3) $z \notin$ dom($MapSubcars_S$)

Rationale. Sometimes the application of the combination pattern may result in a too flat model. In these cases it is possible to 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.

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 tools. We have combined them in the compound model by defining a parent subcharacteristic called *Internal Security*.

Drag-and-drop pattern

Precondition:

1) p, x are two subcharacteristics, $\{p, x\} \subseteq QM_1$.

2) exists a subcharacteristic $r \in QM_S$ such that p $\in MapSubcars_S(r)$.

3) $u \notin QM_S$.

Pattern application: Drag-and-drop(p, x, r, u)

Postcondition:

1) $u \in QM_S$ such that parent(u) = r.

2) $MapSubcars_S(u) = \{QM_1::x\}$.

Rationale. There are subcharacteristics in a component quality model that are not related in any way to subcharacteristics of other component models. In this case the subcharacteristics may be kept as they are in the compound quality model, below a subcharacteristic of the new model.

Example. The subcharacteristic *Spam Filtering Transparency* in the anti-spam domain is kept as it is in the compound model, since it is an interesting subcharacteristic for the new system.

Abstraction pattern

Precondition:

1) p, x is a subcharacteristic, $\{p, x\} \subseteq QM_1$.

2) exists a subcharacteristic $r \in QM_S$ such that $p \in MapSubcars_S(r)$.

3) z, $u \notin QM_S$.

Pattern application: Abstraction(p, x, r, z, u)

Postcondition:

1) z, $u \in QM_S$ such that parent(z) = r, parent(u) = z.

2) $MapSubcars_S(u) = \{QM_1::x\}$.

3) $z \notin$ dom($MapSubcars_S$)

Rationale. Sometimes, when incorporating one or more subcharacteristics of a component quality model into the compound quality model, it may be considered convenient to introduce a new one grouping those subcharacteristics for leveraging the model.

Example. Consider the *Security* subcharacteristic *Safe Communication* from mail servers. We have made an abstraction of this subcharacteristic in the compound model by defining a parent subcharacteristic called *External Security*, since one subcharacteristic *Internal Security* already exists in the compound model.

System pattern

Precondition:

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

2) $z \notin QM_S$.

Pattern application: System(r, z)

Postcondition:

1) z $\in QM_S$ such that parent(z) = r.

2) z $\notin$ dom($MapSubcars_S$)

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

Example. In any compound model it will usually 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.

Fig. 1 shows a final example in the part of suitability that makes use of four of the six patterns. The identification pattern (not shown explicitly in the figure) is applied 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 Management* 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.

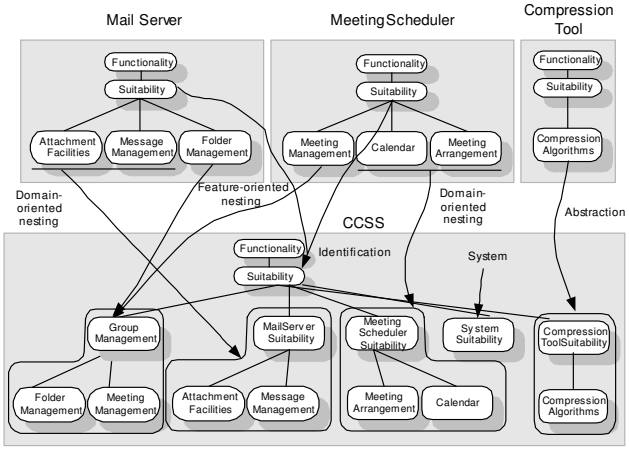

Figure 1. Example of subcharacteristic patterns uses

The patterns defined in this section may be applied in different ways. The strategy we followed in the mail server case can be defined as follows:

- The identification pattern was used in the upper levels of the subcharacteristic hierarchy.
- The combination and nesting patterns were applied in the intermediate levels of the subcharacteristic hierarchy, using one or the other depending on the interest to add a subcharacteristic to aid on structuring the compound model. In the case of the application of the feature-oriented nesting pattern, the added subcharacteristics corresponded to a new concept identified in the compound model.
- The drag-and-drop and abstraction patterns were used in the lower levels of the subcharacteristic

hierarchy, using one or the other depending on the interest to add a subcharacteristic to aid on structuring the compound model.

- The system pattern was used to group every new quality element belonging to a first-level subcharacteristic.

With this strategy, pattern application is a process that could be automated at some extent, except for the last case. This systematic nature facilitates prospective tool-support that is currently being considered.

5.2 Composition patterns for attributes

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 the same 6 patterns than those for subcharacteristics. Their formal definition is slightly different because in these patterns quality elements can be not only classified as attributes but also as subcharacteristics and also because it is necessary to define the metrics of quality attributes (denoted by *Met(attribute)*) in the compound model in relation to the metrics of quality attributes in the component models (see Table 5). However, the changes are so few that we do no include here their definition, except for the case of the identification pattern, since it presents two variants. For the rest of the patterns we just give the rationale and examples of two of them.

Table 5. Attribute patterns.

Pattern	QM_1	QM_2	QM_S	Metrics
Identification	p x	q y	r z	Met(z) = Met(x) Met(z) = K
Combination	p x	q y	r u v	Met(u) = Met(x) Met(v) = Met(y)
Nesting	p x	q y	r z u v	Met(z) = f(u,v) Met(u) = Met(x) Met(v) = Met(y)
Drag-and-drop	p x	N/A	r u	Met(u) = Met(x)
Abstraction	p x	N/A	r z u	Met(z) = f(u) Met(u) = Met(x)
System	N/A	N/A	r z	Met(z) = K

Identification pattern

Precondition:

1) p and q are two subcharacteristics, $p \in QM_1$, $q \in QM_2$, $QM_1 \neq QM_2$.

2) x and y are two attributes, $x \in QM_1$, $y \in QM_2$.

3) exists a subcharacteristic $r \in QM_S$ such that $\{p, q\} \subseteq MapSubcars_S(r)$.

4) $z \notin QM_S$.

Pattern application: Identification(p, q, x, y, r, z)

Postcondition:

1) $z \in QM_S$ such that parent(z) = r.

2) $MapAttrs_S(z) = \{QM_1::x, QM_2::y\}$.

3) The metrics of z may be defined in two different ways:

3.a) Delegation. Met(z) = Met($QM_1::x$).

3.b) Redefinition. Met(z) = K.

Rationale. Concerning a particular quality attribute, it may be the case that one of the component domains prevails over the rest. Therefore, the attribute in the compound model is defined in the same way that the prevalent one (case 3.a, delegation).

On the other hand it is possible that none of the definitions of the attributes of the component model is the appropriate for the attribute in the compound model. In this case this attribute is defined by means of a new metrics (case 3.b, redefinition).

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; thus the metrics of *CCSS::EoLF* will the same as the one of *MailServer::EoLF*.

An example of redefinition may be the *Time in the Market* attribute belonging to the *Maturity* subcharacteristic. Each component model will include this attribute, and the compound one will too, but the value of this last attribute is obviously not computed in the same way that any of the others.

Nesting pattern

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 that system quality attributes depend on the attributes of their components.

Examples. Many examples exist, with different combination functions *f*.

- Some combination functions are additive, remarkably the sum. For instance, the *Total Average Time to Send a Message* (attribute belonging to the *Time Behavior* subcharacteristic) is defined as the sum of 5 attributes: *Average Sending Time*, *Virus Scanning*, *Mail Compression*, *Time to Find Destination Node*, and *Time to Update Log File*. These attributes are a mapping of 5 attributes of the component quality models

corresponding to four domains: mail servers, anti-virus, data compression and routing.

- When putting together quality models, sometimes we are interested in combination functions as 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 values of the attributes that are a mapping of the attributes *Mean Time Between Failures* of every component model.
- Some quality attributes 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 is the *Languages Of Documentation* attribute from the *Understandability* subcharacteristic, defined as the intersection of the attributes that are a mapping of the component models' attributes.

System pattern

Rationale. This 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. Usually this pattern is used in combination with the system composition subcharacteristic pattern.

Example. 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.

6. 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. Firstly, it supports return on investment through model reuse. Secondly, their construction can be requirements-driven and therefore, individual quality models may be left incomplete. Thirdly, 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. Last, tool-support is possible and in fact we currently have a first prototype for building individual quality models and building repositories of information that is the basis for implementing also the patterns presented in the paper [13]. Last it should be remarked that quality models are used either explicitly or implicitly in any trustable CCSS development project; in other words, we claim that COSTUME is not increasing the cost of quality assessment, it just provides a rationale to deal with this activity.

6.1 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 project which the model is being constructed for. Therefore, we consider an iterative selection process [4] 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, when referring to anti-virus tools, 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.

6.2 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 the development of different CCSS of the same type. 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.
- 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*, knowledge 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 [14] 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 easily redefine the activity 3 of COSTUME to take profit of the repository.

6.3 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 did not provide some functionalities, such as meeting scheduling or anti-virus services. 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 these new models.

Also we remark that CCSS models obtained with COSTUME 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.

6.4 Final Remarks

We have defined a precise method for complex quality model construction. Some other methods (not many) exist [6, 7], including our IQMC [10], but they are not specifically oriented for the kind of composite system addressed in the paper. Precision of the method strongly relies on pattern application.

Our definition of CCSS quality model is dual, in the sense that it is highly structured, by keeping track of component models, but it is also ISO/IEC 9126-1-compliant, which can make it more attractive for practitioners.

We would like to enumerate the contexts that may benefit of the existence of quality models: *COTS selection* using the quality model as a framework where requirements and COTS evaluations can be ported into; *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.

Acknowledgments

This work has been partially supported by the CICYT project TIC2001-2165. Gemma Grau work has been partially supported by an UPC scholarship.

References

[1] D. Carney, F. Long. "What Do You Mean by COTS? Finally a Useful Answer". *IEEE Software*, 17(2), March 2000.

[2] A. Finkelstein, G. Spanoudakis, M. Ryan. "Software Package Requirements and Procurement". In *Procs. 8th IEEE* Int. Workshop on Software Specifications & Design, 1996.

[3] J. Kontyo. "A Case Study in Applying a Systematic Method for COTS Selection". In *Procs. 18th IEEE International Conference on Software Engineering*, 1996.

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

[5] *ISO Standard 8402: Quality management and quality assurance-Vocabulary*, 1986.

[6] R.G. Dromey. "Cornering the Chimera". *IEEE Software*, 13(1), 1996.

[7] B. Kitchenham, S.L. Pfleeger. "Software Quality: the Elusive Target". *IEEE Software*, 13(1), 1996.

[8] P. Clements, L. Northrop. *Software Product Lines: Practices and Patterns*. SEI series in SE, Addison-Wesley 2002.

[9] M. Papazoglou, G. Schlageter (eds.). *Cooperative Information Systems: Trends & Directions*. Academic Press, 1998.

[10] X. Franch, J.P. Carvallo. "Using Quality Models in Software Package Selection". *IEEE Software*, 20(1), 2003.

[11] Annie I. Antón. "Goal-Based Requirements Analysis". In *Procs.* 2nd IEEE International Conference on Requirements Engineering, 1996.

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

[13] G. Grau, J.P. Carvallo, X. Franch, C. Quer. "DesCOTS: A Software System for Selecting COTS Components". In Procs. 30th IEEE Euromicro Conference, 2004.

[14] J.P. Carvallo, X. Franch, C. Quer, M. Torchiano. "Characterization of a Taxonomy for Business Applications and the Relationships among them". In *Procs. 3rd Int. Conf. on COTS-Based Software Systems*, LNCS 2959, available at http://www.springerlink.com, 2004.