

# Improving Quality Model Construction Through Knowledge Reuse

Juan Pablo Carvallo<sup>1</sup>, Xavier Franch<sup>2</sup>, Carme Quer<sup>2</sup>

<sup>1</sup>Universidad de Cuenca, Ecuador  
pablo.carvallo@ucuenca.edu.ec

<sup>2</sup>Universitat Politècnica de Catalunya-Barcelona Tech (UPC)  
{franch, cquer}@essi.upc.edu

**Abstract.** Software quality models provide a framework to measure and evaluate software quality of software systems. They are the basis upon which classify requirements and may be eventually used to guide the quantification of these requirements, especially non-functional requirements. Lots of approaches for building quality models have been proposed in the last decades, but still their reuse along different projects is a challenge. In this paper we present several types of knowledge repositories and reuse processes to bridge this gap. The approach implements the idea of software factory and uses some well-known standards and notations like ISO/IEC 25010 as quality standard and the *i\** framework to codify knowledge patterns. We will illustrate how this reuse-based approach helps in obtaining composite quality models for systems that integrate several software components with an individual quality model each.

## 1 Introduction

A quality model (QM) is “the set of characteristics and the relationships between them which provide the basis for specifying quality requirements and evaluating quality” [1]. A QM provides a taxonomy of software quality factors and also metrics for evaluating the quality of a software system used later e.g. in requirements elicitation. Once available, requirements over the system may be stated with respect to the QM.

Lots of approaches for building QMs have been proposed in the last decades [2-10], but still their reuse along different projects is a challenge. In this paper we present several types of knowledge repositories and reuse processes to bridge this gap. The approach implements Basili et al.’s vision of software factory [11] and uses some well-known standards and notations like the ISO/IEC 25010 [1] as quality standard and the *i\** framework [12] to represent system context and codify knowledge patterns, among other things. Specifically, we will illustrate how this reuse-based approach helps in obtaining QMs for systems that integrate several software components for which individual QMs have been constructed.

The rest of the paper is structured as follows. Section 2 presents the required background. Section 3 introduces the artefacts that we have defined to support knowledge reuse in software quality models construction. Section 4 presents the knowledge reuse cycles defined for each of these repositories. Section 5 discusses the related work and finally Section 6 gives some conclusions.

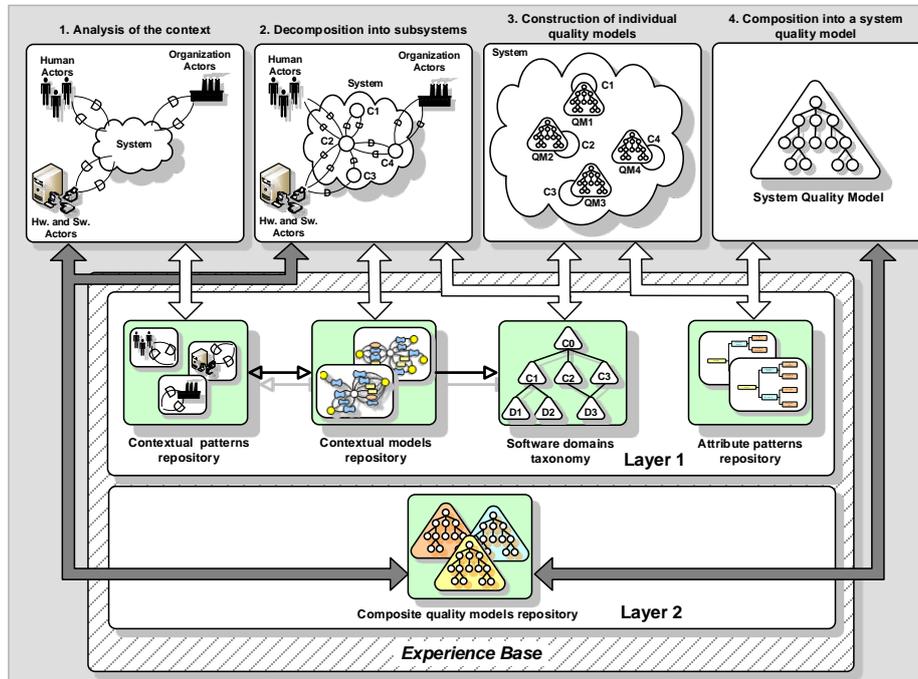
## 2 Quality Models for Composite Software Systems

We describe below the activities necessary for the construction of a QM for a software system that integrate software components, which we call *composite software system* in the rest of the paper. These systems are characterized by: the embracement of distinct functionalities, which are not covered by a single type of component and their need of general-purpose components as for instance anti-virus and compression tools, directory services, etc; This compositional nature requires an approach to QM construction different than the usual monolithic one. In [13][14] we have explored this issue and proposed four activities that conform the resulting QM construction process (see the top part of Fig. 1):

- **Activity 1.** Analyzing the context of the system. The organizational elements that surround the system are identified, as well as other external software systems which the system interacts with. Relationships among the system and the context are established. We propose to model the result of this activity as an *i\** *Strategic Dependency* (SD) model, where there is a distinguished actor representing the system, and a set of contextual actors with which the system maintains one or more relationship dependencies.
- **Activity 2.** Decomposing the system into subsystems. The system is decomposed into several subsystems each offering well-defined services and with a well-defined goal. The subsystems are identified with the help of the results coming from Activity 1. We propose to model the result of this activity as an *i\** SD model that decomposes the one obtained in Activity 1, in which the system actor has been decomposed in several actors (one per subsystem) and the dependencies with the context have been reconsidered and assigned or decomposed to dependencies regarding subsystems.
- **Activity 3.** Building individual QMs for software components that could cover the services and the goal of each subsystem. In this activity an existent method for the construction of QMs for components of a specific software domain has to be applied. In our case, we propose to use the IQMC method that facilitates the construction of an ISO/IEC 25010-compliant QM [15]. The result of this activity is a set of individual QMs.
- **Activity 4.** By composing the individual QMs according to [16], it is possible to arrive to the QM of the system. The result is an ISO/IEC 25010-compliant QM for the whole system that gives a single and uniform vision of the system quality.

## 3 Knowledge Repositories

According to Basili et al. [11] “*improving the software process and product requires the continual accumulation of evaluated experiences (learning) in a form that can be effectively understood and modified (experience models) into a repository of integrated experience models (experience base) that can be accessed and modified to meet the needs of the current project (reuse)*”. The development of this notion resulted in what the authors call the experience factory (see Fig. 2): a logical and/or physical infrastructure aimed at the storage and reuse of all sorts of knowledge (experience and products) resulting from the activities performed in software lifecycle.



**Fig. 1.** Activities and knowledge repositories

In Fig. 2 we may see that the project organization and the experience factory are two separated entities, each with its own mission; the first one is intended to develop and deliver software, whilst the second one is thought to supply on demand the knowledge available from previous experiences. The experience factory processes the information provided by the project organization (e.g. product development and contextual characteristics, data and diverse models), and returns direct feedback of knowledge relevant to each project activity. The experience factory is also responsible for packaging and storing this information in an experience base, from where it can be later accessed, making it available to future projects. In the proposed cyclic approach, the *experience base* is in continuous growth to accommodate new and/or update knowledge obtained as new experiences are performed.

The logical separation of the project development cycle (performed by the project organization) from the systematic learning and packaging of reusable experiences (performed by the experience factory), requires some feedback mechanisms to be implemented in order for these processes to communicate and support each other. To achieve these purposes, two feedback cycles have been defined:

- The *project feedback cycle* (control cycle) is the feedback that is provided to the project during its execution phase.
- The *corporate feedback cycle* (capitalization cycle) is the feedback that is provided to the organization with the main purpose of accumulating reusable experience in the form of software artefacts that are applicable to other projects and are, in general, improved based on the performed analysis.

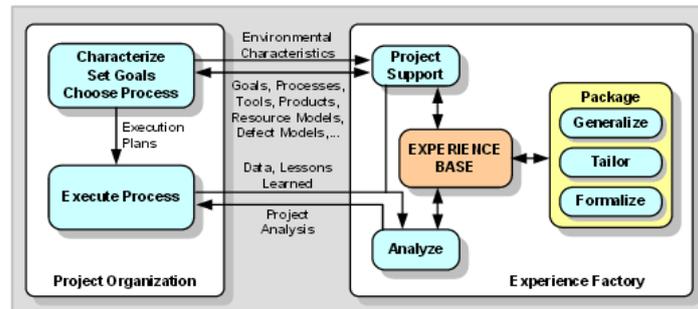


Fig. 2. The experience factory

The QM construction process presented in this paper also aims at improving the return on investment by organizing and reusing the knowledge available from previous experiences. Five artifacts have been identified to support knowledge reuse:

- A *contextual patterns repository* which reflects the most usual interactions among systems and their context.
- A *contextual models repository* which contains contextual models defined in past projects for the software components that compose the target systems.
- A *software domains taxonomy* that organizes QMs built in past projects for the domains of the subsystems in which the systems are decomposed.
- An *attribute patterns repository* which contains quality attributes that can be reused as new QMs are constructed.
- A *composite quality models repository* which contains system QMs, for being reused in case new projects that address a system with the same goals could be applied.

The existence of these repositories and the way in which they interact with the activities largely resemble Basili et al.'s experience factory. On the one hand, the five repositories define an experience base, which makes the knowledge gathered in past experiences available for the support of new projects (control cycle). On the other hand, each of the activities of the produces a set of deliverables which can be packaged and stored in the experience base for their further reuse (capitalization cycle). The process activities interact with the experience base at different levels, either to gather the specific knowledge relevant for them, or to update/extend the knowledge stored with the new deliverables produced. Fig. 1 depicts by means of arrows the existing synergies among activities and repositories. The next sections present each repository in the experience base except for the composite quality models repository that will be introduced in the description of Activity 4 in Section 4.

### 3.1 Contextual Patterns Repository

This repository is used in Activity 1, during the identification of the contextual actors and their dependency relationships with the system actor. For each pattern, the quality factors of the ISO/IEC 25010 that apply may be stated.

Patterns are described using the style of Gamma et al. [17]:

- The *problem* solved by a pattern is expressed as a set of high-level requirements, which will be goals for the system if the pattern is considered.
- The *context* is the same for every pattern in the catalogue, since all of them are patterns to apply in the analysis of the context of a system.
- The *solution* will be described as an *i\** SD model. The pattern provides a scheme of a general solution that must be specialized for its application in a system.
- *Consequences* are non-functional quality factors (which we represent as ISO 25010 subcharacteristics) affected positively or negatively by the use of the pattern.

Fig. 3 shows some examples of patterns, interesting for multiple systems. Let's consider the *Full Availability* pattern. We would select this pattern from the repository in case the composite system considered has the requirement "The system must offer full availability". It identifies two contextual actors needed to provide full availability to a system, namely a system administrator and a system user, and the dependencies among these actors and the system. The *System User* depends on the *System* to obtain the *Full Availability* softgoal. On the other hand, the *System* depends on the *System Administrator* for executing the *Recovery From the Scratch* in case the system fails. The consequences of applying the pattern are the improvement of the fault tolerance and the recoverability of the system-to-be ('+' sign), while harming performance ('-' sign).

A contextual model where these patterns were used is in the contextual model of a Mail Server system (see Fig. 5). The three patterns of Fig. 3 were retrieved and used. The *Tool Administrator* actors and some of the dependencies of the system with this actor were identified thanks to the *Easy Administration* and *Fine Tuning* patterns. Other contextual actors and dependencies were identified from several sources of information that could be reviewed to support this process (e.g. organizational charts).

### 3.2 Contextual Models Repository

There are subsystems with well-defined services and goals that appear over and over in different systems. The contextual models of the software components that may cover the functionality of the subsystems are the models in this repository, and can be reused in the construction of new system QMs where these components are required.

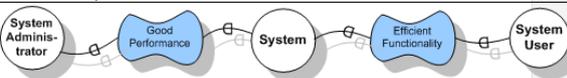
<b>Name</b>	Full Availability
<b>Problem</b>	The system shall offer full availability
<b>Consequences</b>	+ fault tolerance; + recoverability; - performance
<b>Solution</b>	
<b>Name</b>	Fine Tuning
<b>Problem</b>	The system shall offer efficiency in some of its functionalities
<b>Consequences</b>	+ time behavior; + resource utilization
<b>Solution</b>	
<b>Name</b>	Easy Administration
<b>Problem</b>	The system shall be easy to administrate
<b>Consequences</b>	+ Operability
<b>Solution</b>	

Fig. 3. Samples of contextual patterns.

This repository is used in Activity 2, during the decomposition of the system in subsystems. Activity 2 ends up with a system model, which includes its internal system actors or subsystems, its contextual actors and the internal (system) and external (contextual) dependencies among them. A methodological hint to improve this process is to deal with each subsystem as an independent system, and to construct a contextual model for each subsystem (following the same techniques than in Activity 1).

Once the contextual models of each subsystem exist, taking into account the dependencies in each model, the decomposition of the contextual model of the system in subsystems and relationships among them with respect to the system context can be done. In Fig. 4 there are two contextual models of two common subsystem: a Routing Tool (RT) and a Directory Service Tool (DS). They were used in the Mail Servers system to enrich the system model being developed (see an excerpt in Fig. 5).

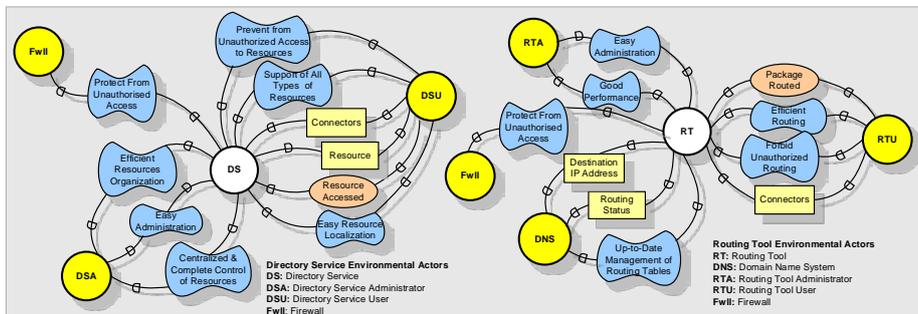


Fig. 4. Samples of contextual models of potential subsystems.

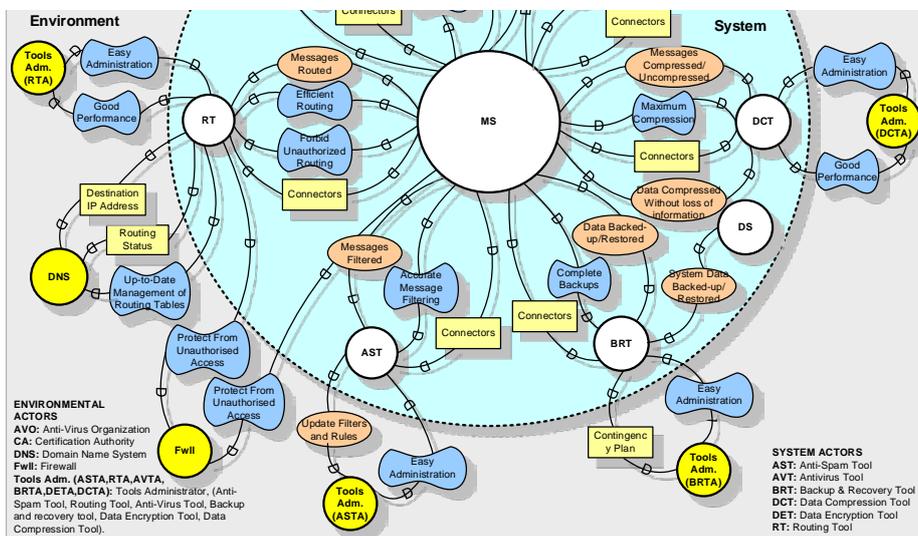


Fig. 5. Excerpt of the Mail Server System model.

### 3.3 Taxonomy of Software Domains

The domains of software components available in the market can be in a repository organized as a taxonomy of scopes, i.e. categories and domains (see Fig. 6).

- Software domains are grouped in the taxonomy into categories that in their turn can be clustered. A category is split into subcategories or domains. Domains appearing in the taxonomy are atomic and may not be further decomposed.
- QMs for each scope are attached to the taxonomy. At the root of the hierarchy, we find the ISO/IEC 25010 QM, whilst the QMs in the rest of scopes contain specialized quality factors specific for components belonging to that category or domain. Quality factors in the QM for a scope are propagated to the QMs of its sub-scopes.

The taxonomy may be used during activities 2 and 3. In Activity 2 it facilitates the identification of subsystem during the decomposition of the system. In Activity 3 it acts as a repository of individual QMs that are already constructed.

Using the goals of a subsystem as search criteria, the taxonomy of software domains is explored. In some cases, complete QMs are found and prepared to be used in Activity 4 during the composition of individual QMs to obtain the system QM. In the worst case, the node of the software domain is not found in the taxonomy or the node of the software domain does not contain a QM constructed during previous projects. If it is not present, the software domain will be included in the taxonomy and the new constructed QM is added to the taxonomy. In order to avoid unnecessary work the QM used as starting point to construct the QM has to be the one in the closest category or domain in the taxonomy, or in the worst case the ISO/IEC 25010 QM.

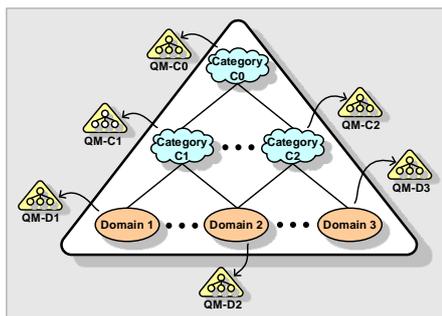


Fig. 6. Software Domains Taxonomy

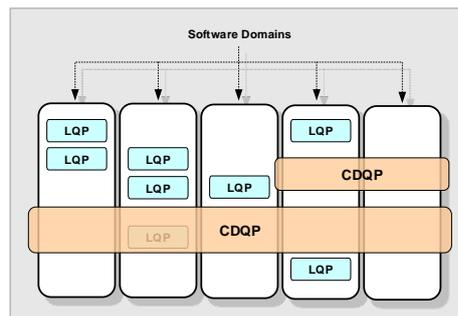


Fig. 7. Attribute Patterns Repository

### 3.4 Attribute Patterns Repository

Our experience in the construction of QMs has revealed that "chunks" of quality factors emerge continuously, either locally when constructing a QM for a particular system actor (*Local Quality Patterns*) or across QMs belonging to different system actors (*Cross Domain Quality Patterns*). We call these chunks *attribute patterns* (see Fig. 7).

These patterns may be used in activities 3 and 4. In Activity 3, they are used if new QMs, not included previously in the taxonomy of domains, are constructed.

**Local Quality Patterns (LQP).** There are quality factors included in the QM of a system actor that share a similar decomposition. The decomposition of one of these quality factors can be made abstract and defined as a LQP. The application of one of these patterns is possible adapting it to different parts of the QM. LQPs include, if necessary, variable labels that are to be replaced by appropriated values each time that they are applied. This happens in QMs of different components, specifically in

stating the existence of functionalities related to the management of some domain object or entity. An example of LQP is a quality factor *entity management* (see Fig. 8) that can be used to decompose the *Functional Completeness* subcharacteristic of a QM. In the QMs of most software domains the functionalities for the management of objects in the domain context is necessary. For instance, in a *Library Loan Management System* this functionality is necessary for the management of books and users. The “entity” variable will be substituted during the application of the pattern by the name of the object class to be managed. The quality factor in the LQP is decomposed into sets of basic attributes representing actions (e.g. addition, update, deletion, etc.), security restrictions (e.g. on the fields that may be updated, the operations allowed, etc.) or even some behavioral settings (e.g. regarding attributes, etc.). In the IP Telephony system, this LQP was applied for stating quality factors about the management of folders and subfolders and agenda structured under the suitability subcharacteristic of the mail server quality model of the Mail Server system.

**Cross Domain Quality Pattern (CDQP).** In other cases, common quality factors can be identified across QMs of different system actors. CDQP may include attributes but also other QM elements such as higher-level (sub) characteristics, the relationships among them and generalised metrics. As a matter of fact, since there are no limits on the number of elements or layers in the patterns hierarchy, they range from simple branches of low-level quality attributes, to whole QMs.

As an example, security attributes such as encryption algorithms, certification systems, security protocols or even system politics are required in many domains, and also were required in the IP Telephony system. Once identified, these CDQP may be reused directly, by incorporating their decomposition into new QMs or as a checklist to identify and/or validate the appropriated attributes required for them. In Fig. 9 the Security Cross Domain Quality Pattern is included.

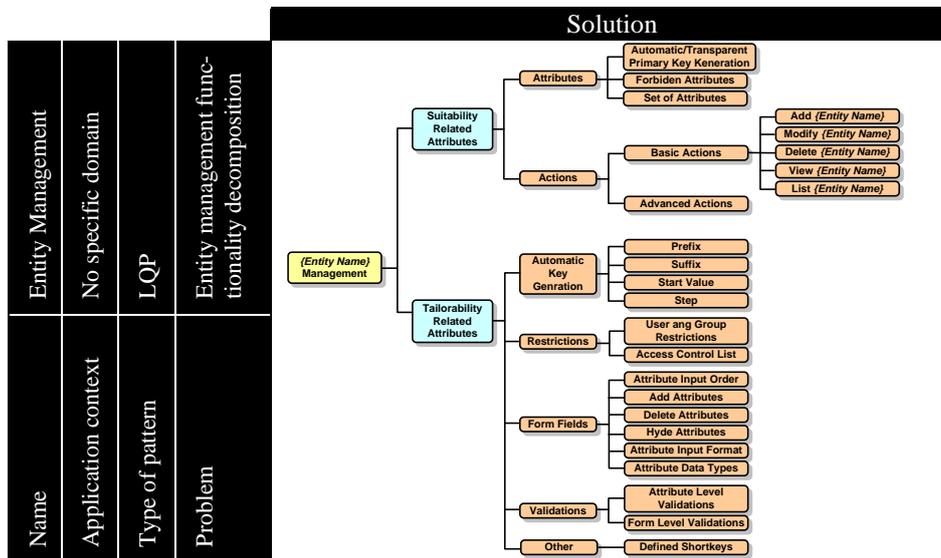


Fig. 8. Sample of the Entity Management Local Quality Pattern

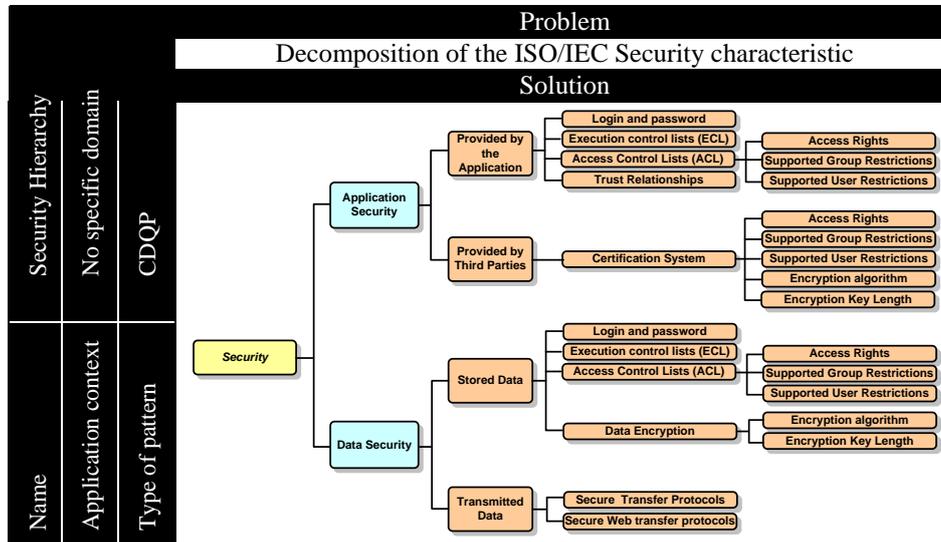


Fig. 9. Sample of the Security Cross Domain Quality Pattern

#### 4 System Quality Model Construction as an Experience Factory

The repositories in the experience base are intended to support all the activities of the construction process (as Fig. 1 shows). Because of this, it describes several feedback cycles. Although each of these cycles involve different repositories and are used for different purposes, they all contain 4 basic phases:

**Phase 1: Exploration (E).** Several information sources (the context of the system, project requirements, deliverables of other activities, etc.) are reviewed and analyzed in relation to the objectives of the activity been conducted. The knowledge gathered and structured in this phase is used as input in the production of the deliverables of the activity (contextual models in Activity 1, system models in Activity 2, system actor's QMs in Activity 3 and the final system QMs in Activity 4).

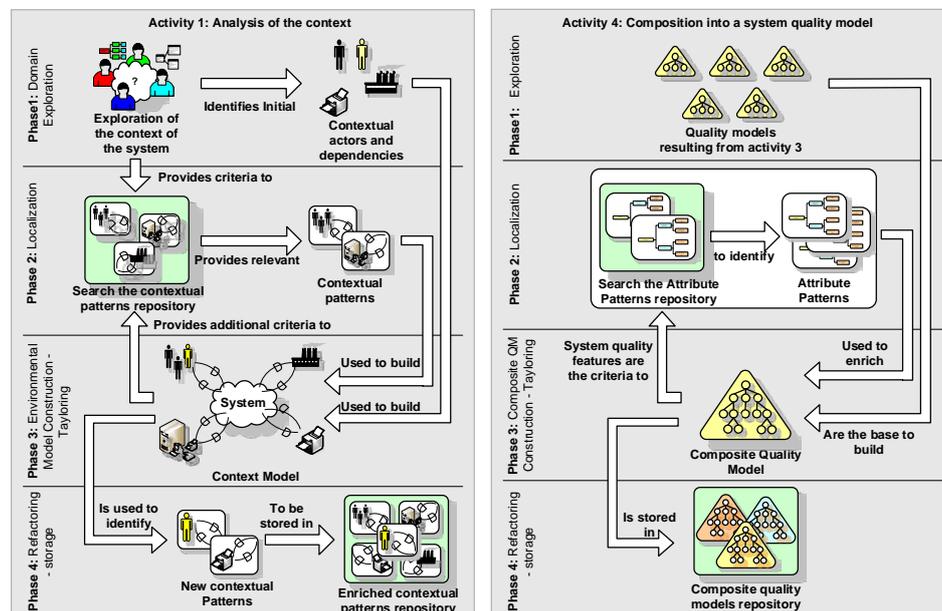
**Phase 2: Localization (L).** The experience base is searched for knowledge relevant for the enrichment and/or production of the deliverables of the activity. The repositories shall be organized in such a way that relevant pieces of knowledge can be easily found and retrieved. Except for Activity 1, the main source for the identification of the localization criteria is the deliverables of the preceding activities.

**Phase 3: Construction-Tailoring (CT).** The deliverables being produced are enriched with the knowledge elements retrieved in Phase 2, taking into account the characteristics of the domain and the type of requirements of the project. Therefore, just those elements related with the requirements of the system will be added. On the other hand, the already-existing elements that do not apply will not be considered.

**Phase 4: Refactoring (R).** For enhancing future reuse, once the deliverables are built, the repositories in the experience base should be updated. The actions to be taken are many: to reorganize the knowledge contained in the repositories, to leave out some pieces of knowledge too constrained to the ongoing project, or even to complete the deliverables beyond the project requirements.

The four proposed phases (figs. 10 to 12) can be combined to define a knowledge reuse cycle for each of the activities of the process:

- **Activity 1.** (*E*) The identification of the initial contextual actors and dependencies relies on several sources of information (e.g. organizational charts, workflow processes, etc.). (*L*) The contextual pattern repository is searched using criteria identified in Exploration (e.g., type of organization). (*CT*) The contextual actors and dependencies identified so far are used to construct the  $i^*$  SD system context model. New information emerging in this process can be used as additional criteria to locate patterns relevant to the case. (*R*) The resulting contextual model is explored for possible recurring situations that can be identified, abstracted as contextual patterns and stored in the contextual patterns repository.



**Fig. 10.** Activities 1 (left) and 4 (right) of the knowledge reuse cycle. The odd ordering of the activities is for convenience of the drawing.

- **Activity 2.** (*E*) Activity 1 can be considered the exploration phase of Activity 2; the contextual model produced in Activity 1 is the departing element used in the construction of the system model in Activity 2. (*L*) This phase includes two search events. First, the internal goals of the system are used as criteria to search the taxonomy and eventually locate the suitable software domains to cover them, that become subsystem actors. Second, contextual models of these subsystem actors are retrieved from the contextual models repository and used to discover new subsystem and/or contextual actors and dependencies that have been previously omitted. (*CT*) Five steps are needed to construct the new  $i^*$  SD model from the one obtained in Activity 1. In them, the needed subsystem actors, identified in the localization phase are used to decompose the system and the dependencies on the contextual actors redefined and or/decomposed regarding these subsystems. (*R*) During decomposition subsystems that do not have a corresponding software domain

in the taxonomy and contextual models repository of the experience base will arise. On the other hand, existing contextual models on that repository will be updated due to changes from its last use. This will result in new software domains and new or updated contextual models that will enrich the experience base.

- **Activity 3.** (E) Activity 2 can be considered the exploration phase of Activity 3. We have also included to this phase the step 0 of the IQMC method [15], aimed at the study and understanding of the software domains of each subsystem actor; this is done constructing a conceptual model which helps to understand each software domain. (L) Using the goals of subsystem actors and the concepts included in conceptual models, the taxonomy is explored to identify departing QMs modelling the quality of each subsystem actor. (CT) The IQMC method is used to tailor the departing QMs obtained in the previous phase and to construct new QMs not found in

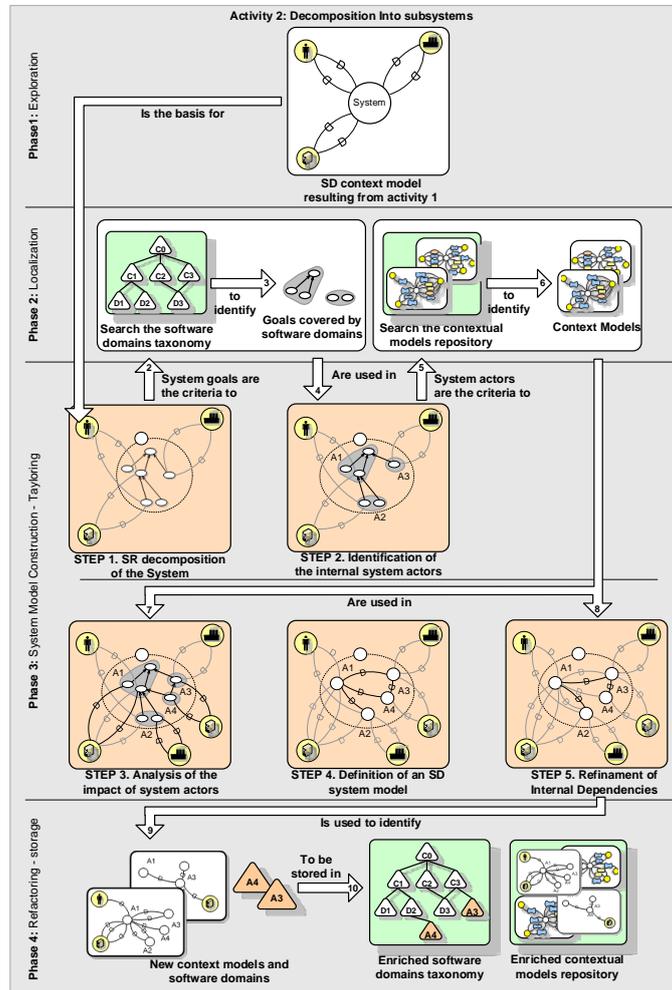


Fig. 11. Activity 2 of the knowledge reuse cycle.

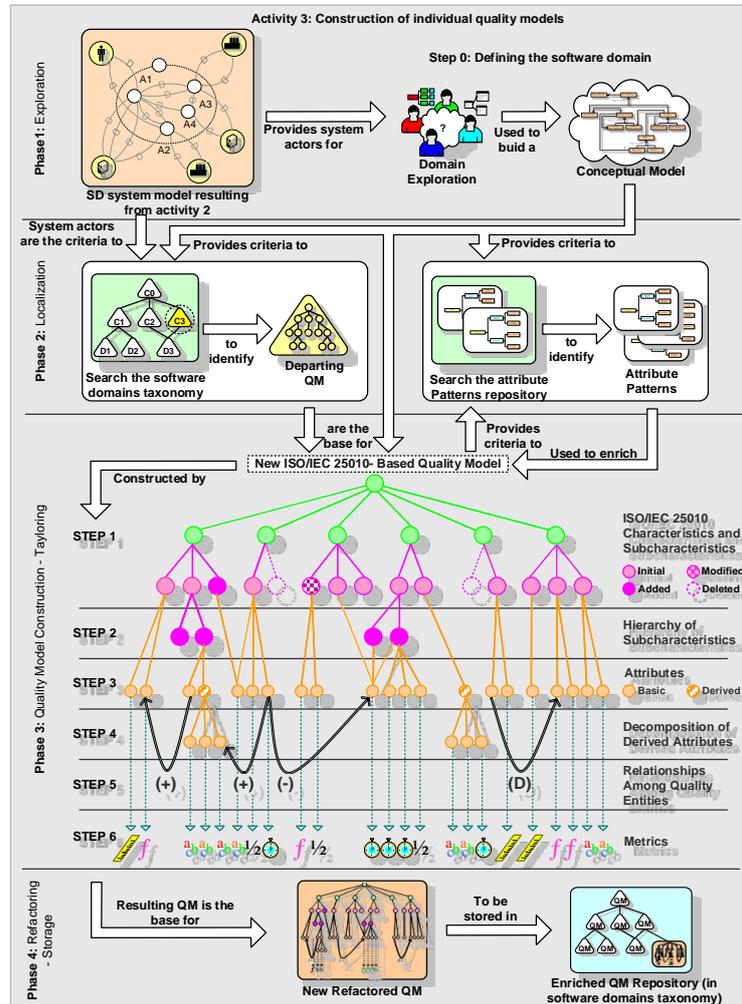


Fig. 12. Activity 3 of the knowledge reuse cycle.

the taxonomy. All the way through this process, relevant attribute patterns retrieved from attribute patterns repository are used to enrich the QM hierarchy with the quality factors included on them. (R) Once defined the individual quality models of subsystem actors, removing context-dependent parts of the resulting QMs in order to obtain generic QMs that can be reused in future experiences allows to add these new QMs or to update existing QMs to be used in next projects.

- **Activity 4.** (E) Activity 3 can be considered the exploration phase of Activity 4. (L) In this phase, attribute patterns relevant for the composite QM being constructed are retrieved from the attribute patterns repository. The main source of criteria to locate these patterns is the quality factors identified during composition of QMs. (CT) By applying the combination patterns introduced in [16], the individual QMs are combined into a system level QM. Relevant attribute patterns identified in the localization phase are used to enrich the resulting QM. The composite QMs is the

final deliverable of the construction process. (*R*) Composite QMs are stored together with the related contextual and system models obtained in Activities 1 and 2 in the composite quality models repository.

Thus, the experience base is structured into two layers. The first one containing the four repositories introduced in Section 3 (Fig. 1, Layer 1), that manage knowledge related to specific software domains, and the second one containing the composite QMs repository (Fig. 1, Layer 2) that manages knowledge at a composite system level. This second layer induces a second reuse cycle where depending on the particular requirements of a new composite system with similar goals, it is possible to use the composite QMs repository as departing repository of system models and composite QMs, that will be used once refactored to meet the particular needs of the new system.

## 5 Related Work

Our work is clearly inspired in Basili's experience factory [11] (see Section 4), and can be considered an application of that approach to quality models construction. Most of the works on quality models, even the most recent ones, focus on quality models elements, structure and properties; as far as we know, none of them propose artifacts and knowledge reuse cycles as the ones presented here. The new existing approaches that mention reuse deal about the adaptation of quality models on different systems or projects [7][8][18] and thus they not provide a holistic version for their construction as done in this paper. Only some of the elements that we present are mentioned in other approaches. For instance, [19][20] include catalogues with decompositions of software quality attributes in relation to security or performance, similar to LQPs and CDQPs introduced in section 3.4, so we have incorporated them. Also some approaches about the definition and use of patterns in  $i^*$  exist. Among them, the closest proposals to ours are the works on social structures presented in [21], where the authors propose a set of social patterns, drawn from research on cooperative and distributed architectures. However, the aim of this work is to propose ontology for information systems, inspired by social and organizational structures. Our work is intended to provide artefacts to support knowledge reuse and improve software quality construction. The scope is distinct, in their work patterns are intended to model different types of cooperation settings among organizations. In our approach, patterns are more detailed and intended to model particular software domains.

## 6 Conclusions

In this paper we have presented a reuse-oriented approach for the construction of quality models (QMs) for composite software systems. This approach was validated first in an academic setting and then used in industrial practices. If we refer to these industrial cases, we have built QMs for 6 domains, e.g. document management tools, workflow tools and IP telephony systems. To give an idea of size, the QM for the IP telephony system case grew up to 1.832 quality factors distributed in a hierarchy of 5 layers and requiring 248 hours of work. The QM combined individual QMs for sub-systems like a directory server, a billing tool, transmission and distribution networks, etc. These numbers and complexity illustrate the need of structure approaches to QM construction as the one we are proposing here. Percentages of reuse of our artifacts

grew up to 80% in the QMs for some of the mentioned system components, and also interestingly enough, up to 40% of the attributes that were introduced in our repositories in the academic validation were reused in all the six industrial cases. At this respect, we defined 31 cross-domain quality patterns and 7 local quality patterns including 36 quality factors. These high percentages are a good indicator of the applicability of our approach, which requires though a more careful validation as future work.

## Acknowledgements

This work has been funded by the Spanish project TIN2013-44641-P.

## References

1. ISO/IEC Standard 25000. *Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE*, 2005.
2. J.A. McCall et al. “Factors in Software Quality”. RADC TR-77-369, 1977.
3. T. Gilb. *Principles of Software Engineering Management*. Addison Wesley, 1988.
4. S. Keller, L. Kahn, R. Panara. “Specifying Software Quality Requirements with Metrics”. *Systems and Software Requirements Engineering - IEEE Tutorial*, 1990.
5. R.G. Dromey. “A Model for Software Product Quality”. *IEEE TSE* 21, 1995.
6. F. Radulovic, R. García-Castro. “Extending Software Quality Models - A Sample in The Domain of Semantic Technologies”. *SEKE* 2011.
7. M. Kläs, C. Lampasona, J. Munch. “Adapting software quality models: Practical challenges, approach, and first empirical results”. *EUROMICRO-SEAA* 2011.
8. C. Lampasona et al. “Software quality modeling experiences at an oil company”. *ESEM* 2012
9. I.J. Jureta, C. Herrensens, S. Faulkner. “A comprehensive quality model for service oriented systems”. *SQJ* 17(1), 2009.
10. M. Oriol, J. Marco, X. Franch. “Quality Models for Web Services: A Systematic Mapping”. *IST* 56(10), 2014.
11. V. Basili, G. Caldeira, H. Rombach. “The Experience Factory”. In *Encyclopedia of Software Engineering*, John Wiley and Sons, 1994.
12. E. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD, 1995.
13. J.P. Carvallo, X. Franch, C. Quer. “Determining Criteria for Selecting Software Components: Lessons Learned”. *IEEE Software*, 24(3), 2007.
14. X. Franch, C. Quer, J.A. Cantón, R. Salietti. “Experience Report on the Construction of Quality Models for Some Content Management Software Domains”. *ICCBSS* 2008.
15. X. Franch, J.P. Carvallo. “Using Quality Models in Software Package Selection”. *IEEE Software* 20(1), 2003.
16. J.P. Carvallo, X. Franch, G. Grau, C. Quer. “COSTUME: A Method for Building Quality Models for Composite COTS-Based Software Systems”. *QSIC* 2004.
17. E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
18. A. Bianchi et al. “Quality models reuse: experimentation on field”. *COMPSAC* 2002.
19. M. Barbacci et al. “Steps in an Architecture Tradeoff Analysis Method: Quality Attribute Models and Analysis”. *CMU/SEI-97-TR-029*, 1997.
20. L. Chung, B. A. Nixon, E. Yu, J. Mylopoulos, *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 1999.
21. A. Fuxman et al. “Information Systems as Social Structures”. *FOIS* 2001.