5 Constructing and Using Software Requirements Patterns

Xavier Franch, Carme Quer, Samuel Renault, Cindy Guerlain, Cristina Palomares

Abstract. Software requirement reuse strategies are necessary to capitalize and reuse knowledge in the requirements engineering phase. The PABRE framework is designed to support requirement reuse through the use of software requirement patterns. It consists of a meta-model that describes the main concepts around the notion of pattern; a method to conduct the elicitation and documentation processes; a catalogue of patterns; and a tool that supports the catalogue's management and use. In this chapter all these elements are presented in detail making emphasis on the construction, use and evolution of software requirement patterns. Furthermore, the chapter includes the construction of a catalogue of non-technical software requirement patterns for illustration purposes.

5.1 Introduction

Requirements elicitation is the process of acquiring system requirements from system stakeholders. The quality of this process is critical to make information technology (IT) projects a success.

When a company runs many elicitation processes over time, it is often the case that a significant proportion of requirements is recurrent and belongs to a relatively small number of categories, especially in the case of non-functional [1] and non-technical [2] requirements. Capitalising on knowledge acquired in previous projects seems in this way an adequate strategy to improve the quality of requirements, and then increase the changes of project success; as well as to increase the efficiency of the requirements elicitation process. This chapter proposes an application of the concept of *software requirement pattern* as a means to capture and capitalise requirements knowledge in the context of IT systems and services procurement projects. Specifically it presents this concept in the mark of the PABRE framework making emphasis on the construction, use and evolution of software requirement patterns.

The chapter is structured as follows. Section 5.2 presents the context of our work. Then in Section 5.3, we summarize the state of the art on software requirement patterns. We present the main elements of our PABRE approach in Section 5.4, and in Section 5.5 we describe the patterns and catalogue structure as well as their construction process. In Section 5.6, we detail our experience in building a catalogue of patterns for non-technical requirements. Finally, Section 5.7 presents some conclusions and future work.

5.2 Context

The work presented in this paper stems from the needs of the Public Research Centre Henri Tudor (TUDOR) at Luxembourg when conducting IT procurement projects over time. Since 2004, TUDOR works in collaboration with freelance and independent consultants. These consultants are federated in a business network that we refer as CASSIS. They are trained to innovative methods produced by research projects and they use these methods in industrial contexts. TUDOR monitors their activity to ensure that they do not deviate over the time. One of the main methodologies delivered to consultants is a requirement engineering method used to design Software Requirements Specification documents (SRS) for IT procurement projects in small and medium size companies [3].

Consultants work in collaboration with customers to help them in identifying their needs for a new IT system supporting their business activities, and then selecting the most relevant system accordingly to their needs. In this particular context, requirements engineers' consultants define SRS for external customers and not for their internal purpose. Consultants' customers are usually looking both for an IT system and for its implementation. In other words, they have requirements towards an IT system and towards additional services. For this reason, the scope of the SRS often encompasses functional, non-functional and non-technical requirements.

The initial goal of the SRS is to serve as a basis for a competitive procurement process. So their primary use is for IT sales managers to understand the needs of the customer and to propose a commercial bid. Only when this process is achieved, the SRS is used in second intend as source for the design or the customization of the selected IT system.

So far, consultants and TUDOR have performed more than 40 projects in compliance with the methodology. The initial approach for capitalising requirements knowledge among the consultants was quite basic. It consisted in re-using fragments of a former SRS as a basis to build the new SRS. This approach was simple to use but required to be aware of the former projects, which was not easy for the consultants due to their decentralized organisation in a business network.

The second TUDOR approach to capitalise requirements knowledge was to design SRS' templates based on existing SRS with similarities. This approach no longer requires the consultants to be aware of all former projects. However, the SRS' templates remained unstructured as domain experts built them both on their own knowledge and on assumptions of similarities found in existing SRS but without any underlying meta-model.

The limitations of these reuse approaches led us to the adoption of a more elaborated framework for requirements reuse.

5.3 Patterns in Requirements Engineering

As in any other software engineering discipline, reuse has been a matter of research in requirements engineering. Reviewing the literature, we may find different approaches for implementing a reuse program within the context described in Section 5.2, i.e. facilitating the process of requirements elicitation and also improving the quality of the resulting SRS. We may classify these approaches depending on: the structure of capitalized knowledge; the language in which the requirements are expressed; the classification and browsing capabilities of the repository; and the existence of a method for building, evolving and exploiting the requirement knowledge repository. From these aspects, in this chapter we focus on the first one, the structure of the capitalized information using patterns.

In the context of engineering, the term "pattern" was introduced by the architect Christopher Alexander that proposed them to improve the quality of the buildings' construction. In his view, *"each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice"* [4]. This formulation is so generic that fitted well in other engineering domains and in particular, software engineers adopted it in several contexts, remarkably related with software design (being software design patterns [5] and software architectural patterns [6] the most representative approaches), but also in other software development phases. In particular, several approaches have proposed the use of patterns as a reuse strategy in the requirements engineering phase, which can be roughly classified as follows:

- Specific pattern-based approaches. We group here those approaches whose patterns cannot be applied in every project but just in those that are compliant to some property. Examples are:
  - Artifact-oriented patterns. Patterns that apply to a particular type of model or diagram. For instance, use case patterns propose use cases to be included in the specification of a system to ensure some properties or achieve some goals [7].
  - Domain-oriented patterns. Based upon the notion of variability proposed in domain engineering. Whilst common requirements are necessary in any system of the domain, other requirements can be chosen or not for a specific system [8]. In some of these proposals, rules are provided to establish dependencies among variable parts of the requirements specifications.
- Refinement-oriented pattern-based approaches. They establish how the attainment of certain goals can be achieved in a certain system. They usually adopt a goal oriented modeling language as *i** [9] or KAOS [10]. Requirements engineers are guided in the process of deciding which requirements are necessary to implement in a system to satisfy certain goals.

- Template-oriented pattern-based approaches. Templates with some additional information about when to use them. The ultimate goal of these approaches is to produce an SRS.
  - In their simplest form, they do not follow any structure, or this structure is very basic even if enriched with some search facilities [11, 12]. In these cases, they promote direct reuse (i.e., copy-and-paste) of templates as requirements, which are written as natural languages sentences usually compliant to a language grammar [13].
  - More elaborated approaches include additional information about the context where they can be applied that guides the requirements engineer during the requirements elicitation process [14, 15]. Usually these proposals are general-purpose in terms of domain although others are specific (e.g. [16, 17] for real-time patterns). Most of them still keep natural language as preferred notation for expressing the requirements, but we may find some that use other notations (e.g., UML [16]) or even combine two (this is the case of [17] that combines natural language with real-time temporal logics).

In the rest of the chapter we present our PABRE template-oriented approach to conduct PAttern-Based Requirements Elicitation. It consists of a meta-model that describes the main concepts around our notion of pattern [18], a method to conduct the elicitation process [19], a catalogue of patterns classified according to some schema, and a tool that supports its management and use [20]. The main result of the application of PABRE is an SRS whose requirements are written in natural language.

5.4 Software Requirement Patterns in PABRE

In this section we describe the notion of *Software Requirement Pattern* (SRP) as used in PABRE. We present the structure of patterns through a meta-model (see Fig. 5.1) and an example, the *Economic Information* pattern (see Fig. 5.2), that illustrates the SRP structure and helps to understand the meta-model behind them.

An SRP is a pattern that, when applied, produces software requirements related to the objective (goal) of that pattern. Giving an analogy with the context-problem-solution Alexander's definition of patterns, goals correspond to *problems* to be solved by applying the SRP. Applying the *Economic Situation* SRP we may produce requirements related to the goal of *Assessing the economic situation of the supplier* that procures a software system.

In our analysis of SRS we have observed that a goal can be achieved in different ways. To deal with this situation, we define an SRP as consisting of several *Forms*, each one representing a different *solution* for achieving the goal. In the *Economic Situation* SRP, its goal can be attained by asking the supplier the relevant economic information (*Economic Situation Information* form), or by setting conditions or prerequisites on the economic situation that the supplier should have (*Economic Situation Prerequisites* form).

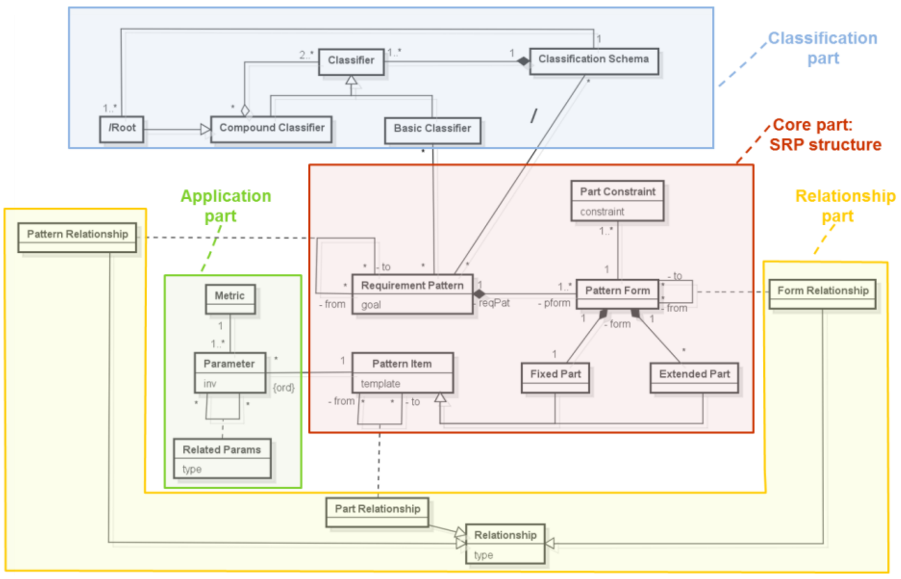
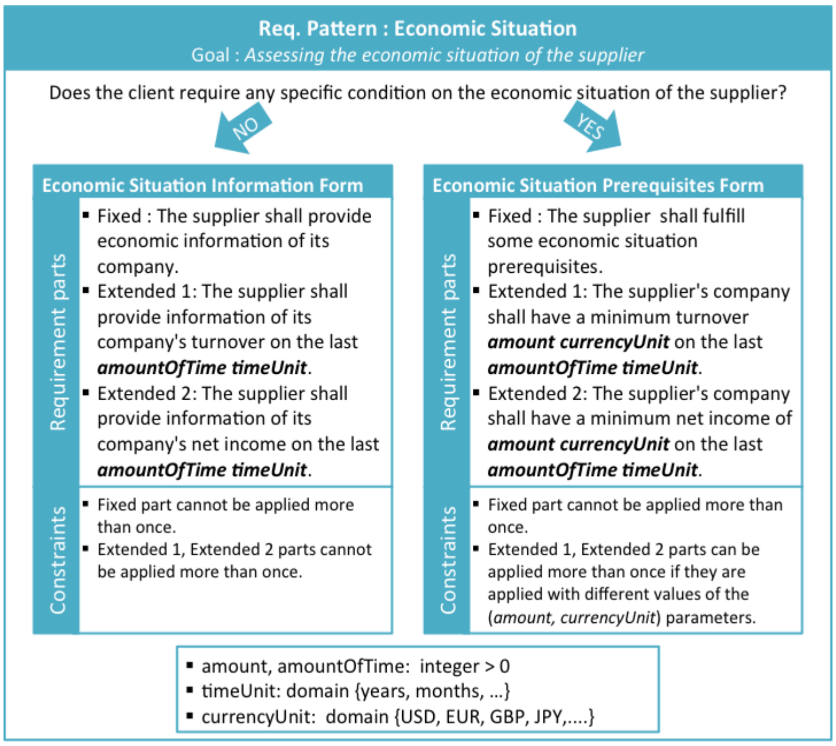


Fig. 5.1: Meta-model for software requirement patterns.


Req. Pattern : Economic Situation

Goal : *Assessing the economic situation of the supplier*

Does the client require any specific condition on the economic situation of the supplier?

NO

YES

Economic Situation Information Form	
Requirement parts	▪ Fixed : The supplier shall provide economic information of its company. ▪ Extended 1: The supplier shall provide information of its company's turnover on the last ***amountOfTime timeUnit***. ▪ Extended 2: The supplier shall provide information of its company's net income on the last ***amountOfTime timeUnit***.
Constraints	▪ Fixed part cannot be applied more than once. ▪ Extended 1, Extended 2 parts cannot be applied more than once.

Economic Situation Prerequisites Form	
Requirement parts	▪ Fixed : The supplier shall fulfill some economic situation prerequisites. ▪ Extended 1: The supplier's company shall have a minimum turnover ***amount currencyUnit*** on the last ***amountOfTime timeUnit***. ▪ Extended 2: The supplier's company shall have a minimum net income of ***amount currencyUnit*** on the last ***amountOfTime timeUnit***.
Constraints	▪ Fixed part cannot be applied more than once. ▪ Extended 1, Extended 2 parts can be applied more than once if they are applied with different values of the (*amount, currencyUnit*) parameters.

▪ amount, amountOfTime: integer > 0
▪ timeUnit: domain {years, months, ...}
▪ currencyUnit: domain {USD, EUR, GBP, JPY,....}

Fig. 5.2: The *Economic Situation* software requirement pattern

Nevertheless, even considering a *Form*, we may find variations in the way they are detailed in different specifications. We have therefore organized a *Form* into *Parts*, each of them being a template. Each *Form* is characterized by a *Fixed Part* which states the minimal requirement that always apply when applying that form, and some *Extended Parts* which may be applied or not in each occurrence in a project.

The *Fixed Part* always becomes a requirement when an SRP is applied with this *Form*. *Extended Parts* are only used if more precise information is required in the specification. Due to this nature, the *Fixed Part* is usually quite generic and hardly measurable. For instance, the first form of *Economic Situation* is *The supplier shall provide economic information of its company*, whilst the two extended parts identify the type of information required (company's turnover or net income) and the period of time.

In general, fixed and extended parts must conform to some *Part Constraint* represented by means of a regular expression that may involve some predefined operators (e.g., for declaring multiplicities or dependencies among parts, as *excludes* and *requires*). In the *Economic Situation* SRP, each part of the forms may be used just once in a specification project, and there are neither *excludes* nor *requires* dependencies among them.

From a syntactic point of view, both fixed and extended parts are similar, therefore an abstract superclass *Pattern Item* is included in the meta-model. Their *templates* are composed by the text to be used as a requirement and optionally some parameters to be instantiated when applying the pattern. Parameters establish their *Metric*, eventually a correctness condition *inv*, and also may be *related* to other parameters (belonging to other patterns) such that they must have the same value. The second form in the *Economic Situation* SRP declares two extended parts that identify additional conditions on this form. For example, the second extended part allows stating prerequisites on the net supplier incomes (by assigning values to the parameters *amount* and *currencyUnit*, e.g. 1M EUR) for a certain period of time (by assigning values to the parameters *number* and *timeUnit*, e.g. 2 years). The metrics of these parameters are detailed at the bottom of the figure.

SRP are not isolated units of knowledge, instead there are several types of relationships among them. In the PABRE approach, we identify three types of relationships:

- *Pattern Relationship*. The most general relationship that implies all the forms and all the forms' parts of the related patterns.
- *Form Relationship*. A relationship at the level of forms implies all the parts of the related forms.
- *Part Relationship*. The relationship only applies to these two parts.

In any case, if *A* is related to *B* and *A* is applied in the current project, the need of applying or avoiding *B* must be explicitly addressed. The types of relationships are not predetermined in the meta-model to make it more flexible. The superclass *Relationship* includes an attribute to classify each relationship.

5.5 A Catalogue for Software Requirement Patterns

The existence of patterns by themselves does not ensure an efficient implementation of requirements reuse. It is necessary to set up an infrastructure able to support the analyst to organize and apply them. In the PABRE framework, we are coping with this aspect through a catalogue of SRP.

5.5.1 Structure of the Catalogue

PABRE's catalogue stores the collection of SRP identified so far. A fundamental issue is the need of classifying them over some criteria for supporting their search. In fact, it is important to observe that different contexts (organizations, projects, standards, etc.) may, and usually do, define or require different *Classification Schemas*. History shows that trying to impose a particular classification schema does not work. For this reason, PABRE decouples SRP from classification schemas (see Fig. 5.3): the latter just impose different structuring schemas on top of the former. SRP are bound to *Basic Classifiers*, whilst *Compound Classifiers* just impose the usual hierarchical structure of any classification schema. Several *Roots* for a classification schema are allowed.

The meta-model (Fig. 5.1) shows that an SRP may be bound to several classification schemas, and even to more than one basic classifier in a single classification schema. In other words, we do not impose unnecessary constraints that could lead to rigidness. For instance, a classification schema may not cover all existing SRP (i.e., some SRP may not be classified).

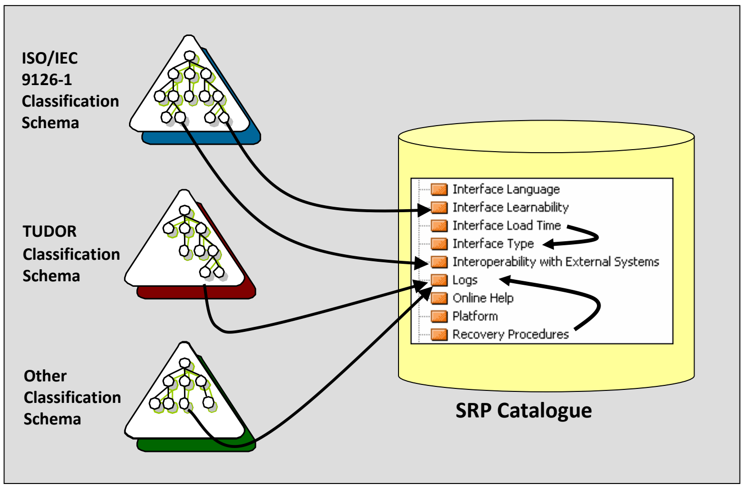

ISO/IEC 9126-1 Classification Schema
TUDOR Classification Schema
Other Classification Schema
Interface Language
Interface Learnability
Interface Load Time
Interface Type
Interoperability with External Systems
Logs
Online Help
Platform
Recovery Procedures
SRP Catalogue

Fig. 5.3: Software requirement patterns classification schemas

5.5.2 SRP catalogue construction

The current PABRE SRP catalogue was built as a result of analyzing the SRS of a certain number of projects in which TUDOR was involved. These SRS are usually broken down into three distinct parts: functional requirements, non-functional requirements (NFR) and non-technical requirements (NTR). Our previous experience in quality models [21] and in requirements engineering projects and the analysis of TUDOR SRS showed us that non-functional requirements and non-technical requirements have higher reuse frequency than functional requirements. Then, our aim for the first version of the catalogue was to represent those SRP whose application leads to NFR that appear in the mentioned SRS [22]. From the experience gained, we recently finished the second version of the catalogue in which we added the SRP corresponding to the NTR, as presented in Section 5.6.

In both cases, the steps (Fig. 5.4) were:

1. Alignment. First, the requirements of the different SRS are consolidated and aligned according to their type. This corresponds to the identification of the departing requirements in the SRS. To make this alignment more reliable it is convenient to identify the concepts addressed by requirements. As part of the process, requirements need to be leveraged, which usually requires decomposing complex requirements into simpler ones. As a result, this step delivers a set of requirement types.
2. Analysis. For each of these types, a study of their adequacy as an SRP is performed. The main criterion of course is repetition that identifies high probability of reuse: those requirements that appear in most or all of the SRS are clear candidates. But this is not the only condition. A requirement appearing in a few, even just one, SRS may also be considered adequate as SRP. In this step, expert assessment is the cornerstone, since experts are the only ones that may say e.g. that a requirement appearing in just one SRS could in fact have appeared in all of them, in other words that its absence is a flaw. As a result, this step restricts the former set to a subset with all the types that may be considered patterns' seed or SRP candidates. The different requirement types are converted into SRP candidates mainly by means of abstraction, but also a consistency analysis and grammatical improvement is applied.
3. Formulation. The selected SRP candidates are converted into SRP. Not every candidate is necessarily converted into a different SRP, since some of them may be considered close enough as to be integrated in the same pattern. As a result, the final structure of the patterns, their forms, their parts and parameters, emerges. In the process, again with expert assessment, the final structure of every SRP may be slightly different than the corresponding requirements in the SRS, since experts may consider that for future projects these differences could be useful. For the templates, syntactical conventions may be enforced.

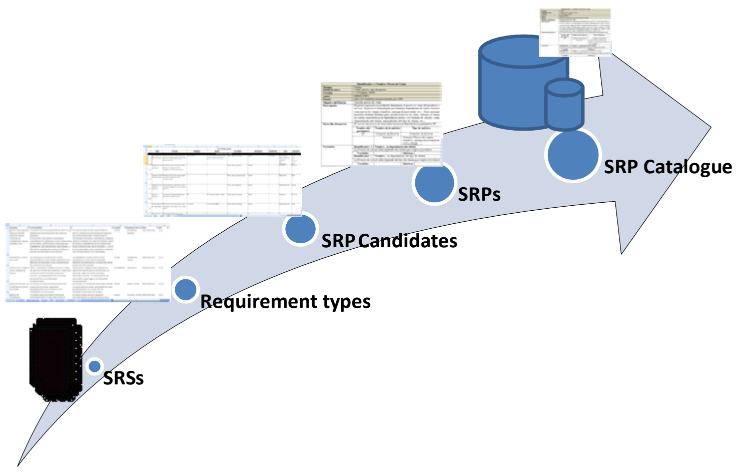


Fig. 5.4: Software requirement patterns catalogue construction process

4. Catalogue construction. Finally, the patterns evolve from individual artifacts into an articulated structure of knowledge, stored in the catalogue. Two things need to be done. First, the SRP need to be classified according to the existing classification schemas. Second, the relationships among SRP are established, as well as those (less frequent) among parameters.

5.5.3 The SRP catalogue use

The SRP catalogue is used during the requirements elicitation phase of IT systems and services procurement projects. During this use, requirements engineers select SRP from the catalogue that apply to the particular project, and converts them into the real requirements that finally configure the SRS. The complete PABRE method is detailed in [7]. In a nutshell, it converts requirements elicitation into a process of search in, and pick-up from, the SRP catalogue (Fig. 5.5).

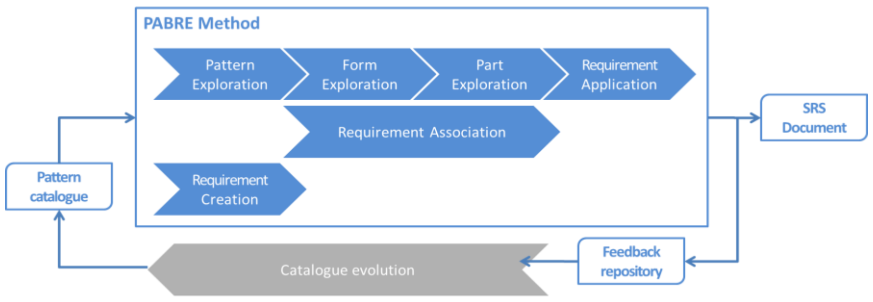


Fig 5.5: Overview of the PABRE method

During elicitation, the catalogue is explored according to the following procedure:

- Pattern Exploration. The requirements engineer selects the next applicable pattern according to some criteria (e.g., the classification schema, the SRP relationships, etc). Based on an explanation and with continuous support from the engineer, the customer decides whether the pattern applies in the project or not.
- Forms Exploration. For each selected pattern, the requirements engineer explains the different forms. Then the customer chooses the form that suits his/her situation and moves to the next step. If no form meets the customer requirements, the requirements engineer elaborates the requirement(s) and moves to the requirement creation step.
- Parts Exploration. For each selected form, the requirements engineer explains the different extended parts. If it is necessary the consultant skims over the parameters and gives example of possible values, in order to improve understanding of the parts. The customer chooses the extended parts that considers necessary for his/her project. As well as in the previous steps, if no extension fits completely into the customer needs, it is necessary to elicit the missing bits separately.

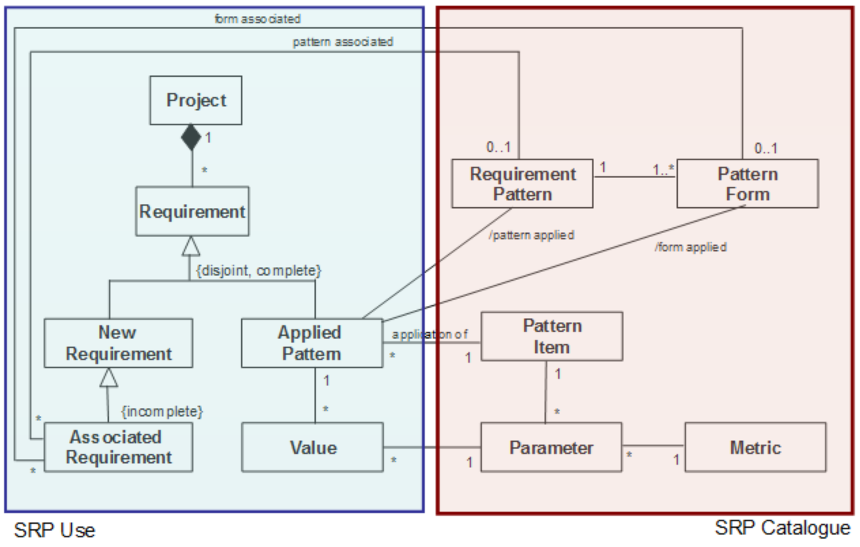


Fig. 5.6: Use of software requirement patterns

At this point, the requirement may be defined in different ways. Fig. 5.6 shows the three types or *Requirement* subclasses and their relationships regarding the SRP meta-model:

- Applied Pattern. For the selected parts, the requirements engineer gives more details about the parameters that apply (e.g., details on possible correctness conditions, dependencies to/from other parameters) and presents the list of values for each parameter. Then the customer chooses the values for the parameters. The requirements engineer turns the customized part(s) into a requirement. The requirements engineer needs to check consistency, dependencies and correctness of the selected parts. When the requirements engineer detects a conflict or an inconsistency, he/she warns the customer and they try to solve the conflict. The resulting requirement is represented with the *Applied Pattern* subclass.

- New Requirement or Associated Requirement. Sometimes, the requirements engineer needs to create a *New Requirement* from scratch because the restriction expressed by the requirement cannot be defined as application of any SRP. We distinguish one particular case: if the new requirement is related with an existent pattern, since it has its same goal, but it is not its direct application, this new requirement is an *Associated Requirement.* An associated requirement consists of partial and small changes of the pattern or the forms (its part's text or parameters).

5.5.4 The SRP catalogue evolution

Catalogue evolution allows capitalizing the different projects and keeping the SRP catalogue up-to-date. The requirements experts identify the patterns, forms, extended parts and parameters which are the most and less used. According to their feedback, different actions can be undertaken to evolve the catalogue.

The feedback is obtained by having the real numbers of SRP applications, the associated requirements to patterns or forms, and the new requirements, over time:

- The number of applications of a pattern versus the number of associations to that pattern can be used by the requirements engineer as a guarantee of the validity of the SRP. If the number of applications is low regarding to its associations maybe the requirements engineer has to check the associated requirements in order to find out if there is some problem with the definition of the requirement. On the other hand, the number of applications is a confirmation of the validity of the pattern.
- The associated requirements have to be analyzed because they can correspond to forms or parts of a pattern that have never been identified before, and that would be helpful for the requirements analysts to have them as parts of the pattern.
- In the case of new requirements, it has to be analyzed if there has been an error in defining them as new, or if in fact the requirement analyst is right and there is not the goal corresponding to the new requirements represented by any SRP of the catalogue. In the first case, the new requirement is analyzed as an associated requirement, and in the second case the new requirement is considered for being added as an SRP following the lasts steps presented in Section 5.2.

5.6 A Software Requirement Patterns Catalogue for Non-technical Requirements

The goal of this section is to illustrate the process of construction of a set of SRP presented in Section 5.2. We describe the construction of the SRP catalogue part corresponding to NTR applicable to TUDOR's projects. NTR are those requirements that do not refer directly to the intrinsic quality of software, but to the context of the system under analysis. They include economic, political and managerial issues. This type of requirements is highly independent of the software domain, and for this reason, good candidate for our work. The complete catalogue of NTR patterns (NT SRP for short) is available in the PABRE website (http://www.upc.edu/gessi/PABRE/index.html).

5.6.1 Preliminaries

We used 6 SRS as starting point of the process, which is distilled next in terms of the different steps enumerated in Section 5.2.

In these 6 SRS documents, specific sections were supposed to contain separately NFR and NTR. However, when building the previous catalogue of SRP for NFR, we discovered that this separation was not clear, since some NTR were discovered in the NFR section. As a result, besides the 29 NFR patterns, we already identified 3 patterns that became the initial set of NT SRP.

The requirements in the SRS were written in French. However, the biggest core of knowledge on requirements engineering is available mainly in English. Also, for dissemination purposes, we had the goal of producing the pattern templates in English too. Therefore, before the alignment process, we translated the requirements into English. The translation was supervised by the TUDOR team since French is their native language but they are also fluent in English.

5.6.2 Alignment

Next, we undertook the alignment looking for requirements expressed differently in each of the SRS but addressing the same concept. Table 5.1, first three rows, shows three requirements appearing in different SRS but related to the concept, namely *Maintenance Period*.

On the other hand, some SRS requirements were also broken into several simple requirements. For instance, the two last rows of Table 5.1 show two requirements that appeared in an SRS as one single complex requirement.

5.6.3 Analysis

In this step it was necessary to consider the requirements that address the same concept to be joined, by means of abstraction, consistency analysis and improvement of the grammatical form, in requirement types.

Table 5.1: Examples of aligned requirements

Concept	Requirement	Keywords
Maintenance Period	The solution should be maintained for three (3) years from the expiration of the warranty period.	Maintenance Period Warranty
Maintenance Period	The proposed solution must be maintained for at least 1 year from the date of expiry of the warranty period.	Maintenance Period Warranty
Maintenance Period	From the date of expiry of the warranty period, the contractor agrees to provide, at the explicit request of the client, ongoing maintenance services for a minimum period of one year.	Maintenance Period Warranty
Audits	The *customer* reserves the right to conduct audits of the provider and its production during the project	Audits Provider Project Production
Audits	These audits will focus on the specific development (product code, development methodology, documentation), the treatment of the reported anomalies and quality procedures.	Audits Specific Development Reported Anomalies Quality Procedures

Table 5.2 shows the list of candidate SRP for the requirements presented above in Table 5.1. The first one corresponds to the abstraction of the three first requirements in Table 5.1. The requirement was abstracted in order to allow the statement of different periods of maintenance after the end of the warranty. This example shows a usual way to implement abstraction, namely substituting specific aspects related to one project by parameters with some associated metric (which of course allows the generation of the abstracted requirements).

Also, some grammatical rules on the SRP templates were enforced. Examples are: requirements were written in an active voice; requirements were written in third-person and with use of the modal verb *shall* suitable for legal requirements or statements.

To ensure catalogue consistency, we built and maintained a glossary of terms and metrics. Since we started from the previous state of the catalogue which contained non-functional SRP, metrics as *timeUnit* and terms as *supplier* and *system* were already therein. This last term was used to substitute the *solution* in the SRS. Also other terms were substituted for the same reasons as *project production* by *project deliverables*.

Also consistency among requirements was checked. For example, we found two requirements at the same SRS: “At each Steering Committee meeting, a statement of progress will be prepared and signed by the parties” and “The report will be prepared by the provider and approved by the customer, if necessary after the required updates” related to the *Steering Committee Meetings* requirements. As can be seen, in the two requirements a different term is used to refer the meeting reports (statement in the first requirement), and inconsistencies among the report approval process are present in them. Therefore clarification was needed to ensure consistency.

Table 5.2: Examples of requirement types

Concept	Requirement	Keywords
Maintenance Period	The supplier shall maintain the system for ***number timeUnit*** from the expiration of the warranty period.	Maintenance Period Warranty
Audits	The customer shall do audits of the supplier or the project deliverables if it is considered necessary.	Audits Supplier Project Deliverables
Audits	The audits shall focus on the ***quality aspects***.	Audits Specific Developments Reported Anomalies Quality Procedures

5.6.4 Formulation

The requirement types corresponding to SRP candidates were processed iteratively, considering at each iteration one type of candidates addressing the same concept. At each iteration, the considered types were compared to the set of the already approved SRP, in order to decide their treatment: approval as a new SRP, incorporation as parts of existing SRP, or discard.

We illustrate the formulation with a particular NT SRP. The requirement types related to *Audits* (see Table 5.2) were included in the catalogue as just one SRP since they address the same concept. When all the SRP candidates aligned to *Audits* were considered, we observed that there are two different groups: one constraining audits for assessing the quality of the supplier in a general way and the other that constraints audits conducted according to a certain quality standard. Therefore, the resulting SRP was structured into two alternative forms: *General Quality Assessment* form and *Quality Standard-based Assessment* form (see Table 5.3). In the first form, the most general requirement type has been selected as the fixed part of the form whilst the other becomes an extended part, since this second type of requirement will not appear in a project without including the first one.

The process above was iterated for the rest of requirement types. Eventually, we found some special situations. On the one hand side, some requirement types were restricting the *Delivered Documents* of the project. When the glossary was browsed, this term was found as the name of an existing non-functional SRP. Therefore, these types were analyzed with respect to this SRP: some types were found redundant regarding to the existent pattern, whilst other were used to constitute a new pattern. The non-functional SRP *Delivered Documents* addresses the statement of requirements on the content of delivered documents, and the NT SRP *Document Characteristics* allows constraining the characteristics of the documents (i.e., their language, electronic format, metadata to include, etc.). Finally both patterns were considered as related to NT aspects, although the first one is also non-functional due to its relationship with the maintenance and understandability of a system and therefore may appear also classified under this perspective.

Table 5.3: *Quality Assessment* non-technical software requirement pattern

QUALITY ASSESSMENT			
Goal: Stating the customer's right of performing quality assessment			
Requirement Form *General Quality Assessment*	**Parts Constraints**	• *Fixed part* cannot be applied more than once. • *Review focus* cannot be applied more than once. • *Quality criteria agreements* cannot be applied more than once.	
	Fixed Part	**Form Text**	If the customer considers it necessary during the system implementation project, s/he shall be allowed to assess the quality of the process or the ***projectDeliverables***.
		Parameters	**Metrics**
		projectDeliverables: is a non-empty set of the different products delivered during the system implementation project	ProjectDeliverables = Set(ProjectDelive-rable) ProjectDeliverable = Domain (hardware, software, data and documents provided or paid by customer as project deliverables, etc.)
	Extended Part *Review Focus*	**Form Text**	The customer shall focus the quality assessment on the ***qualityAspects***.
		Parameters	**Metrics**
		qualityAspects: is a non-empty set of the different quality aspects to be assessed	QualityAspects = Set (QualityAspect) QualityAspect = Domain (specific development, treatment of the reported abnormalities, quality procedures, etc.)
	Extended Part *Quality Criteria Agreement*	**Form Text**	The customer shall agree with the supplier the level of quality expected for the various project deliverables.
Requirement Form *Quality Standard-based Assessment*	**Parts Constraints**	• *Fixed part* cannot be applied more than once. • *Process Quality Assessment* cannot be applied more than once • *Deliverables Quality Assessment* can be applied more than once, only if it is applied for different values of the *projectDeliverables* and *qualityStandard* parameters • *Quality criteria agreement* cannot be applied more than once. • *Quality criteria establishment* cannot be applied more than once.	
	Fixed Part	**Form Text**	If the customer considers it necessary during the system implementation project, s/he shall be allowed to assess the quality of the process or project deliverables taking into account a quality standard.
	Extended Part Process Quality Assessment	**Form Text**	The quality of the process shall be assessed taking into account the ***qualityStandard*** quality standard.

		Parameters	**Metrics**
		qualityStandard: represents the identifier of the quality standard that shall be used to assess the quality	QualityStandard = Domain (IEEE830, IEEE829, IEEE1016, ISO/IEC9126, ISO/IEC 15504-5, etc.)
	Extended Part Deliverables Quality Assessment	**Form Text**	The quality of the ***projectDeliverables*** shall be assessed taking into account the ***qualityStandard*** quality standard.
		Parameters	**Metrics**
		projectDeliverables as above	ProjectDeliverables as above
		qualityStandard as above	QualityStandard as above
	Extended Part *Quality Criteria Agreement*	**Form Text**	The customer shall agree with the supplier on the level of quality expected for the project deliverables.
	Extended Part *Quality Criteria Applied*	**Form Text**	The customer shall establish the subset quality standard criteria to be applied ***timePreposition date***.
		Parameters	**Metrics**
		timePreposition represents the relationship with respect to a date	TimePreposition = Domain (on, before, after, at, by,…)
		date: is a time point representing the date in which the quality standard criteria shall be established	Date = TimePoint

On the other hand, some of the requirement types dealt with one restriction on the concept *Source Code*. Specifically they were about the need of documenting the source code. In this case, they were added as extended parts of the already existent *Source Code* NT SRP.

As already mentioned, during this step and the previous one, expert assessment was crucial. Validation was done by requirement engineers from TUDOR with wide experience in requirements elicitation. Some relevant observations follow.

First of all, the experts provided a general observation about the focus of the forms. For instance, for those SRP referring to suppliers, most were asking for information about the supplier, instead of restricting how the supplier should be or should behave. They proposed to formulate improved forms of the SRP in a more prescriptive way. For instance, this was done in the case of the SRP *Supplier Workforce*, whose goal was initially formulated as “Having information about the supplier workforce” and whose only form’s fixed part was “The supplier shall provide workforce information about the company”. After the expert’s assessment, the goal was transformed into “Assessing the workforce of the supplier” and a new form was added establishing a restriction of the supplier workforce with the fixed part "The supplier shall fulfill some workforce requirements". Both forms have extended parts to establish different aspects of the workforce information to obtain or to restrict respectively.

Experts also suggested restructuring some SRP while iterations progressed. Examples of actions are: SRP merged during the process due to redundancy; extended parts upgraded into fixed parts; even reallocation of extended parts from one SRP to another. For instance, the *Installation* SRP was subsumed by the *Implementation Planning* SRP, since in this SRP it is already established the planning of the different activities, being installation just a particular case. Also, changes in the vocabulary and abstraction from specific contexts of application were continuously performed. For instance, in the case of the SRP about *Audits*, the experts suggested to change in the SRP body the action "audit" by "assess of the quality".

After the validation step we have arrived to 38 NT SRP.

5.6.5 Catalogue construction

The created NT SRP were stored in the PABRE catalogue. As already mentioned, the catalogue already contained three NT SRP identified in the previous version of the catalogue: *Help Desk*, *Crash Response* and *Source Code Documentation*.

The NT SRP were classified in terms of the two classification schemas incorporated into PABRE so far: the ISO/IEC 9126-1 standard [23] and the classification schema defined by the TUDOR center. In this section we illustrate the classification using the ISO/IEC 9126-1 standard.

ISO/IEC 9126-1 does not include non-technical features. However, in previous works we enlarged this standard with NT features [21] and we use this extension (called NT-ISO/IEC 9126) in the PABRE catalogue, which adds 3 characteristics (*Supplier*, *Business* and *Product*) and 15 subcharacteristics to the standard. Before classifying the NT SRP according to this schema, some changes had to be done to take into account some differences on the use of the catalogue.

On the one hand, we found during the process of classification we found 19 patterns that did not correspond to any subcharacteristic in NT-ISO/IEC 9126. The reason is that initially that catalogue was created to include the criteria to assess the quality of a final software product, whereas the NT SRP state requisites for the procurement of a system (probably by gluing or adapting several products). This is the reason why we needed to add a new characteristic to group the SRP about the implementation project: the *Project* characteristic, decomposed into two subcharacteristics: *Business Scheduling* and *Supplier Relationships*.

Table 5.4: NT SRP Extended ISO/IEC 9126-1 classification

1. Supplier	*NT SRP*	*NT SRP Goals*
1.1 Organizational Structure	• Supplier Administrative Information	*Being able to contact the supplier*
	• Supplier Organization	*Understanding the supplier's organization*
	• Supplier History	*Being aware of the history of the supplier company*
1.2 Positioning and	• Supplier Economic Information	*Assessing the economic situation of the supplier*

Strength	• Supplier Workforce	*Assessing the workforce of the supplier*
1.3 Reputation	• Supplier Business Experience	*Assessing project's experience*
	• Supplier Quality Certification	*Assessing quality certification of the supplier*
1.4 Services Offered	• Training	*Stating the training the supplier shall provide about the implemented system*
1.5 Support	• Maintenance Procedure	*Assessing the supplier's maintenance procedures*
	• Type of Maintenance	*Stating the specific types of maintenance for the system implemented the supplier shall provide*
2. Business		
2.1 *Licensing* Schema	• Source Code Licenses	*Stating the source code licenses*
2.2 Ownership	• Intellectual Property Rights	*Stating the rights of using assets result of the project*
2.3 Guarantees	• Warranty	*Stating the warranty that shall be applied over the implemented system*
2.4 Costs	• Cost Breakdown Structure	*Stating the structure of the global cost of the system to be implemented*
2. ***Project***		
2.8 Business Scheduling	• System Implementation Scheduling	*Stating the scheduling of the system implementation*
	• Project Progress Control	*Having or stating the indicators for assessing the progress of the project*
	• Project Management Method	*Stating the method used for project management*
	• Final acceptance	*Stating the time and conditions for the final acceptance of the implemented system*
	• Release	*Stating the time and conditions when the implemented system shall be released*
	• Analysis Stage Activities	*Stating the activities to take during analysis stage*
	• Data Migration	*Stating the necessity of migrating data*
	• Development Activities	*Stating the activities to take during development stage*
	• Acceptance Tests	*Stating the type of tests for the system implementation acceptance*
2.9 Supplier Relationships	• Steering Committee	*Stating the steering committee organization*
	• Meetings Organization	*Stating system implementation meetings organization*
	• Access to Customer Premises	*Stating the rules for supplier access to customer premises*
	• Privacy	*Stating the privacy rules among customer and supplier*
	• Project Progress Control	*Having or stating the indicators for assessing the progress of the project*
	• Quality Assessment	*Stating the customer's right of performing quality assessment*
	• Payment Procedure	*Stating the payment schedule*
	• Settlement of Disputes	*Stating how the disputes between customer and supplier shall be solved*
	• Supplier People Assigned to the Project	*Assessing the profile of the people assigned to the project*
	• Help Desk	*Having access to a technical support service for the system for information and assistance*
	• Crash Response	*Stating the required level of service for supplier support in case of crash*
3. Product		
3.1 History	• Products History	*Assessing the history of the main products that will be part of system to be implemented*
	• Community Support	*Assessing the existence of a community that could give support on the implemented system*
3.2 Deliverables	• Delivered Documents	*Stating the documentation that shall be delivered*
	• Source Code Documentation	*Stating the source code licenses*
3.3 Parameterization and Customization	------------------------------	

On the other hand, some related subcharacteristics were merged into just one. Specifically, they were those related to the cost of the business. The original subcharacteristics were too static: *Licensing Costs*, *Platform Costs*, *Implement Costs* and *Network Costs*, but the new subcharacteristic integrates all these costs in a cost breakdown structure allowing the flexibility to add new ones.

Also relationships among the SRP were investigated. With this aim, we took into account the keywords stated for each SRP (obtained during their construction), and also the metrics of the parameters of the different SRPs. For the *Quality Assessment* SRP, taking into account the parameter *ProjectDeliverables* (Table 5.3), we identified a dependency with the *Delivered Documents* SRP that also has a parameter with the same metrics. The relationship is that the documents for which a quality assessment is done must be deliverable documents.

In Table 5.4, the 37 SRP are classified taking into account the extended NT ISO classification schema updated to include the new identified characteristics and subcharacteristics.

5.7 Conclusions

In this chapter we have presented the PABRE framework for reusing requirements knowledge following a pattern-based approach. The different components of PABRE have been introduced: its meta-model, the processes supported and the catalogue of patterns. For illustration purposes, we have described the construction of the first version of a set of 37 non-technical requirements patterns that follow the structure stated in the PABRE meta-model. Requirements engineering experts from the TUDOR research center have been collaborating in this construction.

Future work spreads over several dimensions.

- Validation of the adequacy of PABRE in other types of IT projects beyond the procurement projects targeted so far.
- Adoption of clear rules and best practices for writing pattern templates (see e.g. EARS [24]).
- Extension of the catalogue with functional patterns from several domains (e.g., in the context of TUDOR, ERP and CRM procurement projects).
- Improving capabilities of tool support by introducing recommendation capabilities (e.g., "projects that used this pattern usually use this other").

In addition, more validation is needed. We have so far conducted post-mortem analysis of the SRS coming from past projects to validate that: the meta-model covers the features expressed in those SRS; the coverage of the catalogue is satisfactory. Still, we need to apply it to real cases in an action-research basis.

Acknowledgements

This work has been partially supported by the Spanish project TIN-2010-19130-C02-01.

References

[1] L. Chung and J.C.S. do Prado Leite. “On Non-Functional Requirements in Software Engineering”, *Conceptual Modeling: Foundations and Applications*, 2009. pp. 363-379.

[2] J.P. Carvallo, X. Franch, C. Quer. “Managing Non-Technical Requirements in COTS Selection”. *IEEE International Requirements Engineering Conference (RE),* 2006.

[3] S. Renault, B. Barafort, E. Dubois, M. Krystkowiak, “Improving SME trust into IT consultancy: a network of certified consultants case study”, *EuroSPI*, 2007.

[4] C. Alexander. The Timeless Way of Building. Oxford University Press US, 1979.

[5] E. Gamma, R. Helm, R. Johnson and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 2000.

[6] F. Buschmann, R. Meunier, H. Rhonert, P. Sommerlad, M. Stal. Pattern-Oriented Software Architecture. A System of Patterns, John Wiley & Sons, 1996.

[7] L. Chung, S. Supakkul. “Capturing and reusing functional and non-functional requirements knowledge: A goal-object pattern approach”, *IEEE International Conference on Information Reuse and Integration (IRI)*, 2006.

[8] M. Mannion, H. Kaindl. “Using parameters and discriminants for product line requirements”, Systems Engineering Journal, 11(1), pp.61-80, February 2008.

[9] S. Supakkul, T. Hill, L. Chung, T.T. Tun, J.C.S.P. Leite. "An NFR Pattern Approach to Dealing with NFRs", *IEEE International Requirements Engineering Conference (RE)*, 2010.

[10] R. Darimont, A. van Lamsweerde. “Formal refinement patterns for goal-driven requirements elaboration”, *ACM Symposium on Foundations of Software Engineering (SIGSOFT)*, 1996.

[11] A. Monzon. “A Practical Approach to Requirements Reuse in Product Families of On-Board Systems”, *IEEE International Requirements Engineering Conference (RE)*, 2008.

[12] E. Hull, K. Jackson, J. Dick. *Requirements Engineering*, Third Edition. Springer, 2010.

[13] K. Watahiki, M. Saeki. “Scenario Patterns Based on Case Grammar Approach”, *IEEE International Symposium on Requirements Engineering (RE)*, 2001

[14] S. Withall, Software Requirement Patterns. Microsoft Press, 2007.

[15] J.A. Toval, J. Nicolás, B. Moros, F. Garcia. “Requirements Reuse for Improving Information Systems Security: A Practitioner's Approach”, *Requirements Engineering*, 6(4), pp.205-219, 2002.

[16] S. Konrad, B. H. C. Cheng. “Requirements Patterns for Embedded Systems”, *IEEE Joint International Conference on Requirements Engineering (RE)*, 2002.

[17] S. Konrad, B.H.C. Cheng. "Real-Time Specification Patterns", *ACM/IEEE International Conference on Software Engineering (ICSE)*, 2005.

[18] X. Franch, C. Palomares, C. Quer, S. Renault, F. DeLazzer. “A Metamodel for Software Requirement Patterns”, *Requirements Engineering: Foundation for Software Quality (REFSQ)*, 2010.

[19] S. Renault, O. Mendez-Bonilla, X. Franch, and C. Quer. “A pattern-based method for building requirements documents in Call-for-tender processes,” *International Journal of Computer Science and Applications*, 6(5), pp. 175-202, 2009.

[20 C. Palomares, C. Quer, X. Franch. "PABRE-Man: Management of a requirement patterns catalogue". *IEEE International Requirements Engineering Conference (RE)*, 2011.

[21] J. P. Carvallo, X. Franch, C. Quer. "Determining Criteria for Selecting Software Components: Lessons Learned". *IEEE Software*, 24(3), pp.84-94, 2007.

[22] S. Renault, O. Mendez-Bonilla, X. Franch, C. Quer. "PABRE: Pattern-based Requirements Elicitation". *International Conference on Research Challenges in Information Science (RCIS),* 2009.

[23] "ISO Standard 9126". *Software Engineering – Product Quality, part 1*. International Organization for Standarization, 2001.

[24] A. Mavin, P. Wilkinson, A. Harwood, M. Novak. Easy Approach to Requirements Syntax (EARS). *IEEE International Symposium on Requirements Engineering (RE)*, 2009.