Expressing Customer Requirements Using Natural Language Requirements Templates and Patterns*

A. DURÁN, B. BERNÁRDEZ, M. TORO, R. CORCHUELO, A. RUIZ, J. PÉREZ
Dpto. de Lenguajes y Sistemas Informáticos
Universidad de Sevilla
Facultad de Informática, Avda. Reina Mercedes s/n, 41012 Sevilla
ESPAÑA (SPAIN)

Abstract: - Expressing customer requirements so they can be understood not only by requirements engineers but also by noncomputer professional customers is not an easy task. Natural language is frequently the usual choice for expressing customer requirements in spite of its well–known problems, but using more formal notations too early is a risky choice that can make requirements impossible to understand for customers and users. In addition, using natural language do not guarantee understanding. Requirements engineers do not usually have good writing skills, and sometimes requirements expressed in natural language are not understood because of the poor way they are written. In this paper, we propose to use requirements templates and patterns to improve requirements expression. We have identified two types of requirements patterns: *linguistic patterns*, which are very used, well–understood, sentences in natural language requirements descriptions that can be parameterized and integrated into templates, and *requirements patterns*, which are generic requirements templates that are found very often during the requirements elicitation process and that can be reused with some adaptation. CSCC'99 Proc. pp.3531-3536

Key–Words: – Requirements Engineering, Requirements Elicitation, Requirements Patterns, Requirements Reuse

1 Introduction

Requirements elicitation is the step of requirements engineering through which customers and users of a software system discover, reveal, articulate, and understand their requirements [8]. In [2], problems of requirements elicitation are classified into three main groups: problems of *scope*, i.e. deciding the boundary of the system; problems of *volatility*, since requirements evolve over time; and problems of *understanding* between the communities involved. The main problem of understanding is expressing the requirements in a form that can be understood not only by requirements engineers but also by noncomputer professional customers and users, something that is not usually addressed by elicitation techniques. For example, in JAD and brainstorming sessions, elicited requirements are supposed to be publicly visible by the participants, but the way those requirements are expressed is not described.

The usual choice for expressing elicited requirements is natural language, since it is frequently the only common language to customers, users and requirements engineers. Problems of natural language are well known, but using more formal notations too early is a risky choice that can make requirements impossible to understand for customers and users.

Sometimes, even using natural language, requirements expressed in natural language are not understood because of how they are written — requirements engineers do not usually have good writing skills.

In this paper, we present requirements templates and patterns that can improve understanding by helping requirements engineers and users to express requirements for information systems using natural language. We have developed requirements templates and identified two types of requirements patterns: *linguistic patterns* (*L–patterns*), which are very used sentences in natural language requirements descriptions that can be parameterized and integrated into templates, and *re-*

*This work is funded by the CICYT project "MENHIR". TIC 97-0593-C05-01, Ministry of Education and Science (Spain)

quirements patterns (*R–patterns*), which are generic requirements templates that are found very often during the requirements elicitation process and that can be reused with some adaptation.

The paper is organized as follows. The requirements engineering process model that will be assumed in the rest of the paper is presented in section 2. In section 3 we present templates and patterns for the three kinds of information systems requirement we have identified. Section 4 concludes by giving some conclusions and pointing out some future work.

2 Requirements Engineering Process Model

The requirements engineering process model, and its associated terminology, shown in Fig. 1 will be assumed for the rest of the paper. This model is partially based in the one proposed in [8]. The meaning of each step is the following:

- **Requirements Elicitation**: requirements are elicited from customers and users using elicitation techniques. The results of this process are the customer requirements, or *C–requirements* as they are called in [1], usually written in natural language. This process and its product are the primary objective of this paper, as highlighted in Fig. 1.

- **Requirements Analysis**: C–requirements are analyzed in order to detect inconsistencies and identify missing requirements, usually by building an object–oriented o structured model. Customer or users with knowledge of modeling techniques can participate in the analysis, as indicated by the dashed arrow in Fig. 1.

- **Requirements Specification**: analyzed C–requirements are recorded in a form suitable for software developers, i.e. they are transformed into *D–requirements* [1]. Other usual product of this process is a prototype of the system to be built, which can be very helpful for validation and elicitation of missing requirements.

- **Requirements Validation**: customers and users must validate the requirements and the prototype, usually leading to elicitation of new requirements. The whole process iterates until all requirements are validated and no more requirements are elicited.

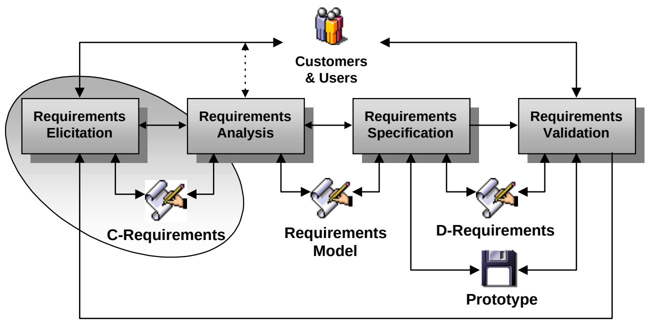

Figure 1: requirements engineering model

3 Requirements Templates and Patterns

Requirements templates help requirements to be expressed and understood. On one hand, requirements information is structured, so requirements engineers know what missing information must be searched, requirements can easily be treated automatically, reuse is promoted, and customers get used to using templates very fast. On the other hand, filling blanks in pre–written sentences, i.e. L–patterns, is easier and faster than writing a whole paragraph saying what the system is expected to do. Moreover, whole requirements templates, i.e. R–patterns, can be reused many times, provided they have been identified, with some adaptation to specific developments.

In next sections, requirements templates and patterns for information systems are described. The used notation is the following: words between < and > must be properly replaced, words between { and } and separated by commas represents options; only one option must be chosen.

3.1 Information Storage Requirements

3.1.1 Information Storage Requirements Template

The most important thing in information systems is information. The template for information storage requirements, see Fig. 2, helps users to answer the question "*what information, relevant for your business goals, must be stored by the system?*". The meaning of the template fields is the following:

- **Identifier and descriptive name**: every requirement must be uniquely identified by a number and a descriptive name. In order to help rapid identification, information storage requirements identifiers start with *RI*.

- **Version**: following IEEE recommendations [5], different versions of requirements must be managed.

This field contains the current version number and date of the requirement.

- **Author, Source**: these fields must contain the name and organization of the author and the source of the current version of the requirement.
- **Purpose**: this field must state *why* the requirement is necessary to achieve business goals.
- **Description**: for information storage requirements, this field uses an L–pattern that must be completed with the *relevant concept* about information must be stored.
- **Specific data**: this field must hold a list of specific data associated with the relevant concept.
- **Time interval**: this field indicates how long information about the concept is relevant for the system. It can takes two values: *past and present* and *present only*. For example, if the concept is *employees*, a *past and present* time interval means that ex–employees are relevant for the system; a *present only* time interval means that ex–employees are not under consideration.
- **Importance, Urgency**: these field indicates how important/urgent the requirement is for customers and users. It can be assigned a numeric value or some enumerated expressions such as *vital*, *important* or *would be nice* for importance or *immediately*, *under pressure* or *can wait* for urgency as proposed in [7].
- **Comments**: other information that cannot be fitted in previous fields can be recorded here.

RI–<*id*>	***<descriptive name>***
Version	<*version number*> (<*version date*>)
Author	<*author*> (<*author's organization*>)
Source	<*source*> (<*source's organization*>)
Purpose	<*purpose of requirement*>
Description	The system shall store the information corresponding to <*relevant concept*>. More precisely:
Specific data	• <*specific data about the relevant concept*> • …
Time interval	{ past and present, only present }
Importance	<*importance of requirement*>
Urgency	<*urgency of requirement*>
Comments	<*comments about the requirement*>

Figure 2: information storage requirements template

An example of use of this template, supposing a video tape renting system, is shown in Fig. 3.

RI–01	**Information about movies**
Version	1.0 (Feb, 17, 1999)
Author	A. Durán (University of Seville)
Source	R. Corchuelo (Super Video Shop)
Purpose	To know availability of movies at any moment and to be able to help customers select a movie using different criteria
Description	The system shall store the information corresponding to movies in the video store. More precisely:
Specific data	• Title of the movie • Number of tapes of the movie rented and ready to rent at any moment • Type of the movie: children, action, science–fiction or adults • Duration of the movie • Main actors of the movie • Director of the movie • Year of production of the movie
Time interval	past and present
Importance	vital
Urgency	immediately
Comments	none

Figure 3: example of information storage requirement

3.1.2 Information Storage R–patterns

After using the templates and patterns described in this paper in several academic projects, we have realized that there are very similar requirements that are present in most developments. For information storage requirements, we have identified some R–patterns such as those referring to information about customers (see Fig. 4), products, orders, invoices, etc. These R–patterns can be easily classified according to different criteria and stored in a repository for further reuse.

RI–x	**Information about <*customers*>**
…	…
Description	The system shall store the information corresponding to <*customers*>. More precisely:
Specific data	• Legal identification number of <*customer*> • Name of <*customer*> • Address of <*customer*> • Telephone numbers of <*customer*> • E–mail address of <*customer*>
…	…

Figure 4: example of information storage R–pattern

3.2 Functional Requirements

3.2.1 Functional Requirements Template

Information systems not only store information, they must also provide services using the information they store. The functional requirements template, see Fig. 5, describes use cases [6], and help users and customers

RF–<*id*>	***<descriptive name>***		
. . .	. . .		
Description	The system shall behave as described in the following use case when <*triggering event*>		
Precondition	<*precondition of use case*>		
Ordinary sequence	**Step**	**Action**	
	. . .	. . .	
	n	{The {<*actor*>, system} <*action performed by actor/system*>, Use case <*use case (RF–x)*> is performed}	
		n.1	If <*condition*>, {the {<*actor*>, system} <*action performed by actor/system*>, use case <*use case (RF–x)*> is performed}
		. . .	. . .
Postcondition	<*postcondition of use case*>		
Exceptions	**Step**	**Action**	
	p	If <*exception condition*>, {the {<*actor*>, system} <*action performed by actor/system*>, use case <*use case (RF–x)*> is performed}, then this use case is {resumed, aborted}	
Performance	**Step**	**Maximum time**	
	q	*m* seconds	
Frequency	This use case is expected to be performed <*number of times*> times/<*time unit*>		
. . .	. . .		

Figure 5: template and L–patterns for functional requirements (use cases)

answer the question "*what do you want the system to do with the stored information in order to achieve your business goals?*". The meaning of the template fields is the following:

- **Identifier and descriptive name**: the same as in information requirements template, except that functional requirements identifiers start with *RF*.

- **Version, Author, Source, Purpose**: the same as in information storage requirements.

- **Description**: for functional requirements, this field contains an L–pattern that must be filled with the *triggering event* that starts the use case.

- **Precondition, Postcondition**: necessary conditions that must hold in order to perform the use case or after normal termination of the use case can be expressed in these fields.

- **Ordinary sequence**: this field holds the ordinary sequence of interactions of the use case. In every step, one actor or the system can perform an action, or other use case can be performed, i.e. *used*, following the semantics of *uses* and *extends* relationships given in [9]. A step can have conditional substeps, assuming that only one substep is performed. Other use cases can be performed in conditional substeps, i.e. the use case can be *extended.*

- **Exceptions**: after performing a step of the use case, some exceptional conditions may arise. This field specifies the behavior of the systems in such circumstances. After the action or the use case associated with the exception (i.e. the *extender* use case) is performed, the system can resume the ordinary sequence or aborts the use case.

- **Performance**: for any step or substep in which an action is performed by the system, a maximum time can be specified in this field.

- **Frequency**: although frequency is not actually a requirement, it is an important information for developers and can be recorded here.

- **Importance, Urgency and Comments**: the same as for information requirements template.

An example of use of this template, supposing the same previous video tape renting system, is shown in Fig. 6.

3.2.2 Functional Requirements R–patterns

For functional requirements, we have identified four R–patterns which are always present in every information system development and that we have named *CRUD* R–patterns (***C****reate*, ***R****ead*, ***U****pdate*, ***D****elete*). These R–patterns must always be present in correct information systems: information stored in the system must be *created* (see Fig. 7), *updated* and *deleted* in order to be synchronized with its environment; In addition, someone must be able to *read* the stored information.

These R–patterns can be easily reused by adding or modifying steps in the ordinary sequence.

RF–07	**Customer returns video tape(s)**		
Version	2.1 (Feb, 10, 1999)		
Author	B. Bernárdez (University of Seville)		
Source	A. Ruiz (Super Video Shop)		
Purpose	To control tape returns and customers payments		
Description	The system shall behave as described in the following use case when a customer wants to return one or more video tapes		
Precondition	all video tapes are Super Video Shop tapes		
Ordinary sequence	**Step**	**Action**	
	1	The clerk requests the system to start the return procedure	
	2	The system requests for tape(s) identification(s)	
	3	The clerk provides all identifications needed	
	4	The system calculates the amount and prints the invoice	
		4.1	If any tape is lately returned, the system charges an extra of 10% for every late return
	5	The customer pays the invoice	
	6	The clerk puts the tape(s) on the shelves	
Postcondition	stored information is updated, tapes are ready to rent again		
Exceptions	**Step**	**Action**	
	3	If some tape is not registered as rented, the system reports the situation to the clerk and does not include the tape in the invoice, then this use case is resumed	
Performance	**Step**	**Maximum time**	
	4	5 seconds	
Frequency	This use case is expected to be performed 50 times/day		
Importance	vital		
Urgency	immediately		
Comments	none		

Figure 6: example of functional requirement (use case)

RF–x	**Create <*new information*>**	
...	...	
Precondition	<*new information*> is not stored yet	
Ordinary sequence	**Step**	**Action**
	1	The <*some actor*> requests the system to start the <*create new information*> procedure
	2	The system requests for <*new information*>
	3	The <*some actor*> provides <*new information*>
Postcondition	<*new information*> is stored	
...	...	

Figure 7: example of functional requirement R–pattern

3.3 Nonfunctional Requirements

3.3.1 Nonfunctional Requirements Template

Other capabilities of the system, such as privacy, reliability, etc. can be expressed using the nonfunctional requirements template. An example can be seen in Fig. 8. This template does not have any specific field, since it is a generic template. The only identified L–pattern, for the moment, is used in the description field and its form is: *The system shall <system capability>*.

3.3.2 Nonfunctional Requirements R–patterns

Not many R–patterns for nonfunctional requirements have been identified for the moment. The example of Fig. 8 could be used as a R–pattern for specifying the operating system under the system must be able to operate if its description field were changed into: *the system shall operate under <operating system>*.

RN–3	**Operating System**
Version	1.0 (Jan, 15, 1999)
Author	A. Durán (University of Seville)
Source	M. Toro (Super Video Shop)
Description	The system shall operate under the Linux Operating System
Importance	vital
Urgency	immediately
Comments	Check different Linux versions compatibility

Figure 8: example of nonfunctional requirement

4 Conclusions and Future Work

In this paper, requirements templates and patterns for information systems have been presented. These templates and patterns can help to express requirements during the requirements elicitation phase of requirements engineering, keeping the benefits from using natural language and avoiding early formalization of requirements. As a result, customers and users *do* understand requirements.

These templates and patterns are currently being successfully used in two real developments of Sadiel S.A., a top software company of Andalucía (Spain), where their use have dramatically improved communication with customers and users.

The idea of using templates for expressing requirements is based in the use case templates by Rumbaugh [9] and, mainly, by Cockburn [3] and his use case template. We have extended Cockburn's ideas to other types of requirement, not only functional requirements, have integrated some L–patterns into the templates and have identified several R–patterns. Inspired by [4], requirements patterns have been a *natural product* of our experience in requirements engineering.

Some possible lines for future work can include adapting templates when more feedback from real developments is available, discovering more patterns, specially for nonfunctional requirements, creating a requirements repository for promoting reuse, and building a CASE tool.

Acknowledgments

We wish to thank Mr. César Pérez–Chirinos, from TransTools S.A., for his *CRUD* acronym and all his time spent discussing about patterns.

References

[1] J. W. Brackett. Software Requirements. Curriculum Module SEI–CM–19–1.2, Software Engineering Institute, Carnegie Mellon University, 1990.

[2] M. G. Christel and K. C. Kang. Issues in Requirements Elicitation. Technical Report CMU/SEI–92–TR–12, Software Engineering Institute, Carnegie Mellon University, 1996.

[3] Alistair Cockburn. Goals and Use Cases. *Journal of Object–Oriented Programming*, Sep–Oct 1997.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Professional Computing Series. Addison-Wesley, 1995.

[5] IEEE. IEEE Guide for Developing System Requirements Specifications. IEEE/ANSI Standard 1233–1996, Institute of Electrical and Electronics Engineers, 1996.

[6] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. *Object–Oriented Software Engineering: A Use Case Driven Approach*. Addison–Wesley, fourth edition, 1993.

[7] IBM OOTC. *Developing Object–Oriented Software: An Experience–Based Approach*. Prentice–Hall, 1996.

[8] S. Raghavan, G. Zelesnik, and G. Ford. Lecture Notes on Requirements Elicitation. Educational Materials CMU/SEI–94–EM–10, Software Engineering Institute, Carnegie Mellon University, 1994.

[9] James Rumbaugh. Getting started: Using use cases to capture requirements. *Journal of Object–Oriented Programming*, September 1994.