Specification Cases: a lightweight approach based on Natural Language

Leandro Antonelli[1], Julio Leite[2], Alejandro Oliveros[3], Gustavo Rossi[1]

[1]Lifia, Fac. de Informática, UNLP, La Plata, Bs As, Argentina
[2]Dep. Informática, PUC-Rio, Gávea, RJ, Brasil
[3]INTEC-UADE, Bs As, Argentina

{lanto,gustavo}@lifia.info.unlp.edu.ar
julio@inf.puc-rio.br
oliveros@gmail.com

Abstract. Use Cases are one of the most used artifacts in software requirements specifications. Although there are a lot of templates that suggests how to describe Use Cases, as well as many quality inspection techniques, there are no many techniques to deal with the complexity and the effort to produce good quality Use Cases. On top of that, Use Cases are biased towards user interaction, leaving out important domain information. Thus, it is necessary to provide techniques to incrementally describe specifications that goes beyond user interaction but which can be derived from simpler artifacts in order to make the process organized and effective. This paper proposes an approach to begin with very simple sentences (kernel sentences), following with the LEL glossary and Scenarios, in order to describe Specification Cases. The approach relies on already defined kernel sentences and suggests three procedures to reorganize the knowledge captured in kernel sentences to describe the LEL, then Scenarios and finally Use Cases. This paper also reports a preliminary evaluation that supports the applicability and usability of the approach.

Keywords: Use Cases, LEL, Scenarios, Kernel Sentences, Requirements.

1 Introduction

Two types of business domains are predominant: (i) very complex where an extensive documentation is necessary, and (ii) simple and well known where some light description is enough. Financial and legal domains are very close to the first one, while market places websites are example of the second one [13] [14]. Use cases are one of the most used artifacts to describe requirements in software development of the first type, while User Stories are mostly used in in developments of the second type.

Use Cases consolidates a lot of knowledge of the domain and the software application that is spread among many people. Some key characteristics of Use Case description are: (i) they should state clearly the limit between the system and the real world, (ii) they should describe the conversation, that is, the interaction between the actor (outside the boundaries of the software system) and the system, without provid-

ing any description of the User Interface of the software system, and (iii) they describes many scenarios (happy path, alternative, exceptional) in only Use case [13]. The techniques used to describe Use Cases are mainly workshop [26] [2] [33] where a big number of stakeholders provide their knowledge and a software professional should understand, evaluate and organize the knowledge in Use Cases. Thus, the professional should cope with a big gap between the stakeholders and the Use Cases. It is necessary to provide some technique to deal with the amount of information in an incremental way, to produce preliminary artifacts until Uses Cases are obtained.

Notwithstanding, the large adoption of Use Cases by industry, this representation lacks a series of important aspects with respect to requirements, among them: the focus on functional requirements disregarding non-functional requirements, focus on the interaction of the future system, without considering the large environment, and failing to consider that the boundary among system and the real world may be blurred, since requirements are evolving artifacts.

Our paper proposes a strategy trying to ameliorate the problems listed above. We are proposing Specification Cases, based on three concepts: kernel sentences, LEL (Language Extended Lexicon), and Scenarios.

The concept of the kernel sentence was introduced in 1957 by linguist Z.S. Harris [19] and featured in the early work of linguist Noam Chomsky [12]. Kernel sentences are also known as basic sentences, declarative construction, in active voice, always affirmative with only one verb. Boyd [9] suggests the use of Kernel Sentences to describe models in Software development. LEL is glossary [27] that has the aim of understanding the language of an application domain without worrying about the application software. LEL categorize terms in four categories (subjects, objects, verbs and states) and uses two attributes (notion and behavioral responses) to describe the terms. Scenarios are description of behavior regarding certain initial context or situation. There are many definitions of Scenarios. This paper will use a Scenario based on Leite conception [27]. These scenarios are naturally linked to the LEL since LEL glossary provides the language to describe the Scenarios, and there are already proposed strategies to obtain Scenarios from LEL [17] [3]. It is important to mention that LEL glossaries and Scenarios do not consider any software application scope, they only describe the application domain.

This paper proposes an approach to use kernel sentences as input to describe the LEL glossary. Then, the LEL is used to describe Scenarios. Finally, multiple Scenarios (sharing the same goal) are combined in one Specification Case. Our proposed approach considers that the kernel sentences are already defined. Eventually, we are developing a technique supported by a tool to describe and validate kernel sentences in a collaborative way. Nevertheless, although kernel sentences are a linguistic concept there is some similar artifact in Software engineering: business rules [18] [29]. Gonçalves [22] propose an approach to obtain business rules from collaborative narrative. The approach proposed does not have the aim to obtain a complete description of a LEL glossary and Scenarios. There are many proposals to obtain good descriptions [4] [5]. This paper proposes a pipeline beginning with kernel sentences, following with LEL, then Scenarios and finally Specification Cases. The aim of this approach is to use the previous artifact of the step in the pipeline to drive the elicitation of more knowledge to describe the following artifact in the pipeline. Thus, this proposal is a framework to describe Specification Cases linking kernel sentences, LEL and Scenar-

ios. The description of Specification Cases from Scenarios requires an important decision. Use Cases describe a software application with its limits, while Scenarios describes an application domain. Moreover, many scenarios should be combined in one Specification Case. Thus, the software developer should analyze many scenarios that share the same goal, and identify the scope of the software, to describe the Specification Case, by bringing the large environment into picture.

It is also important to mention that our proposed approach could also be used to obtain User Stories [14]. User Stories and Uses Cases both recognize a limit between application domain and software. Nevertheless, User Stories are simpler than Use Cases, since User Stories only consider one scenario of interaction. Thus, we believe that our proposed approach can be used to obtain User Stories, although we have to perform some case study to assess our claim.

The rest of the paper is organized in the following way. Section 2 describes some preliminary knowledge needed to understand the approach. Section 3 describes the proposed approach. Section 4 provides evidence about the applicability and usability of the approach. Section 5 discuses some related works. Finally, section 6 presents some conclusion and future work.

2 Background

This section describes basic concepts of kernel sentences, LEL, and Scenarios.

2.1 Kernel Sentences

A kernel sentence is a simple construction with only one verb. It is also active, positive and declarative. This basic sentence does not contain any mood. It is termed as "kernel" since it is the basis upon which other more complex sentences are formed. For example, Fig. 1 describes two kernel sentences. It is important to mention that the verb to be does not have a semantic meaning that is why the second example has two verbs: to water and to be. Fig. 2 shows two sentences that are not kernel, since both sentences has two verbs. First sentence has verbs to fertilize and to add, while second sentence has verbs to water and to prevent. The first sentence can be rewritten in two kernel sentences (**Fig. 3**).

The farmer fertilizes the tomatoes
The farmer waters the tomatoes when it is hot

Fig. 1. Kernel sentences

The farmer fertilizes the tomatoes to add nutrient that are not present in the soil.
The farmer waters the tomatoes to prevent them of drying out.

Fig. 2. No kernel sentences

The farmer fertilizes the tomatoes
The fertilization adds nutrient to the soil

Fig. 3. Sentence rewritten as kernel sentences

2.2 Language Extended Lexicon

The Language Extended Lexicon (LEL) is a glossary that describes the language of an application domain, where not necessarily there is a definition of a software application. LEL is tied to a simple idea: "understand the language of a problem without worrying about the problem" [27]. The language is captured through symbols that can be terms of short expressions. They are defined through two attributes: notion and behavioral responses. Notion describes the denotation, i.e. the intrinsic and substantial characteristics of the symbol, while behavioral responses describe symbol connotation, i.e. the relationship between the term being described and other terms (Fig. 4). Each symbol of the LEL belongs to one of four categories: subject, object, verb or state. This categorization guides and assists the requirements engineer during the description of the attributes. Table 1 shows each category with its characteristics and guidelines to describe them.

Category: symbol
Notion: description
Behavioral responses:
Behavioral response 1
Behavioral response 2

Fig. 4. Template to describe a LEL symbol

Table 1. Template to describe LEL symbols according to its category

Category	Notion	Behavioral Responses
Subject	Who is he?	What does he do?
Object	What is it?	What actions does it receive?
Verb	What goal does it pursue?	How is the goal achieved?
State	What situation does it represent?	What other situations can be reached?

2.3 Scenarios

The Scenarios are description of the dynamic (behavior) of a domain, where not necessarily there is a definition of a software application. It is based on the LEL, since the LEL captures the language (concepts) while the scenarios capture the dynamic (activities). The Scenario describes a sequence of steps (episodes) from some starting point (context) to achieve some objective (goal). Some active agents perform the action (actors) using some objects (resources). Fig. 5 summarizes the template of the Scenario.

Scenario title: id
Goal: objective
Context: starting point: time, place, activities previously achieved.
Actors: active agents
Resources: passive elements: materials, data.
Episodes: List of actions, simple breakdown with no conditions, no iterations.

Fig. 5. Template to describe a Scenario

3 Approach

This section describes the proposed approach in a general way, and after that describes every sub step.

3.1 The approach in a nutshell

The approach is basically a succession of three steps: (i) description of the LEL, (ii) description of the Scenarios, and (iii) description of the Use Cases. Kernel sentences are the input of the approach and Use Cases are the output. Every step uses the product of the previous step as input to reuse some knowledge to describe the product that the step focuses on. Fig. 6 summarizes the approach.

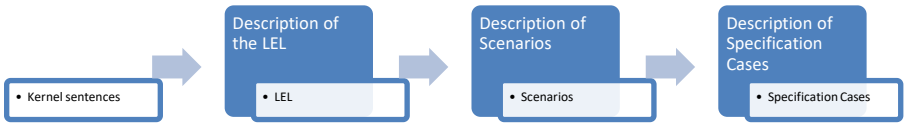

Fig. 6. Our approach in a nutshell

It is important to emphasize some aspects of the approach. Kernel sentences are the input of the approach and their description are outside of the boundaries of the proposed approach. Kernel sentences are simple and basic sentences that can be described in a workshop, a brain storming session or any other collaborative technique.

The approach does not provide guidelines to describe completely the products (LEL, Scenarios and Use Cases). The approach provides guidelines to reuse some knowledge from the previous artifact to perform a preliminary and partial description. Thus, the procedures proposed to describe LEL, Scenarios and Use Cases include some step considering the description relying on the knowledge of the analysist to enrich the description. These steps could be found in the procedure as “add additional…” or “complete manually…” or “improve…”. The approach is a framework for use the knowledge of the previous artifact in the sequence, but this knowledge should be enriched with the knowledge of the requirements engineer.

Kernel Sentences, LEL and Scenarios describe the application domain without considering any software application, while Specification Cases describes a software application. Thus, the step concerning the description of Use Cases should deal with the definition of the scope of the software system. That is, considering the scenarios of the application domain, the requirements engineers should decide what behavior of the application domain would be automatized in the software system.

3.2 LEL description

Kernel sentences are simple expressions following the structure: noun + verb + complement. Thus, the noun is a possible subject in LEL, verb is a symbol of verb category in LEL, and the complement generally has a noun that can be an object in LEL. Then, kernel sentences can also be related with symbols. For example, behavioral responses of a subject describe the actions that he performs. Thus, the kernel sentenc-

es that have certain subject must be behavioral responses of him. Moreover, verbs symbols describe activities, and the behavioral responses describe a set of subtask to perform the activity. Thus, the requirements engineer should analyze the activities to (verbs) to organize them. **Fig. 7** summarizes the algorithm.

```
for each kernel sentence k
   extract noun ni in k
   if ni not defined in LEL
       define ni
       complete the notion manually
   add k to ni.behavioral responses
   extract verb vi in k
   if vi not defined in LEL
       define vi
       complete the notion manually
for each kernel sentence k
   find a verb vi where k is a subtask
   add k to vi.behavioral responses
for each vi defined in LEL
   arrange the behavioral responses in sequence
   add additional behavioral responses
```

Fig. 7. Procedure for LEL description

Let's considers five kernel sentences related to the fertilization process (Fig. 8). According to them, the following symbols should be defined: (i) subject farmer, (ii) objects spraying backpack, irrigation pipe, mixture of minerals, (iii) verbs fertilizes using the spraying backpack, fertilizes using the irrigation pipe, prepares the mixture of minerals, pour the mixture into the irrigation pipe, pour the mixture into the spraying backpack. It is important to mention that verbs symbols of the LEL do not exclusive refer to verbs in the grammar sense. They refer to expression that denote an action. Farmer is the common noun of the all kernel sentences, thus, it should be defined as a subject symbol (Fig. 9). The first 3 behavioral responses are obtained from the kernel sentences, while the rest are added. Then, two examples of verbs are shown: "fertilize using the irrigation pipe" (Fig. 10) and "fertilize using the spraying backpack" (Fig. 11). In both cases, first and second behavioral responses are kernel sentences, while the rest is added.

3.3 Scenarios description

One verb should give origin to one scenario. That is, scenarios should describe a unique flow of actions from the context to the goal. If there are some conditions that give origin to different flows of actions, should be defined two Scenarios. The name of the verb is used as a name of the scenario. The notion of the verbs describes the goal of the scenario (Table 1). And the behavioral responses of the verb describe the episodes of the Scenarios (Table 1). The actor of the scenarios can be derived as the subject symbol who performs the action. And the resources can be obtained collecting the objects of the episodes. Then, the context should be described manually. And the episodes should be enriched in order to describe with more detail the activity. Fig. 12 summarizes the algorithm.

The farmer fertilizes using the spraying backpack.
The farmer fertilizes using the irrigation pipe.
The farmer prepares the mixture of minerals.
The farmer pours the mixture into the irrigation pipe.
The farmer pours the mixture into the spraying backpack

Fig. 8. Kernel Sentences related to fertilization process

Subject: farmer
Notion: responsible to grow the fruits
Behavioral responses
The farmer fertilizes spraying.
The farmer fertilizes watering.
The farmer prepares the mixture of minerals.
The farmer dilutes the mixture in water.
The farmer sprays the mixture to the plant.
The farmer waters the mixture to the soil.

Fig. 9. Symbol subject farmer description

Verb: fertilize using the irrigation pipe
Notion: activity that pursue the aim of adding nutrient to the plant.
Behavioral responses:
The farmer prepares the mixture of minerals.
The farmer pours the mixture into the irrigation pipe.
The farmer pours fresh water into the irrigation pipe.

Fig. 10. Description of the verb symbol "Fertilize using the irrigation pipe"

Verb: fertilize using the spraying backpack
Notion: activity that pursue the aim of adding nutrient to the plant.
Behavioral responses:
The farmer prepares the mixture of minerals.
The farmer pours the mixture into the backpack.
The farmer sprays the mixture to the plant.
The farmer washes the backup.

Fig. 11. Description of the verb symbol "Fertilize using the spraying backpack"

```
for each verb v in LEL
    define a Scenario s
         s.id = v.id
         s.actor = subject containing v in behavioral resp.
         s.goal = v.notion
         s.episodes = v.behav responses
         complete s.context manually
         s.resources = objects in s.episodes
         improve the description of s.episodes
```

Fig. 12. Procedure for Scenario description

The verbs “fertilize using the irrigation pipe” (Fig. 10) and “fertilize using the spraying backpack” (Fig. 11) describe two different ways of fertilization. Every one describes a unique flow of actions and should give origin to one scenario. The scenario “fertilize using the irrigation pipe” (Fig. 13) has 5 episodes. The first and second ones (“The farmer calculates the amount of minerals”, “The farmer dilutes the minerals in water”) are defined considering the first behavioral response of the verb (“The farmer prepares the mixture of minerals”). The third and fifth episodes (“The farmer pours the mixture into the irrigation pipe”, “The farmer pours fresh water into the irrigation pipe”) are obtained from the verb. While fourth episode (“The farmer activates the irrigation pipe”) is added. The scenario “fertilize using the spraying backpack” is obtained in a similar way (Fig. 14).

Scenario: fertilize using the irrigation pipe
Goal: add nutrient to the plant
Context: the cistern has enough water to activate the irrigation pipe
Actor: Farmer
Resource: cistern with water, irrigation pipe, minerals, chart to calculate the amount of minerals.
Episodes:
The farmer calculates the amount of minerals
The farmer dilutes the minerals in water
The farmer pours the mixture into the irrigation pipe
The farmer activates the irrigation pipe
The farmer pours fresh water into the irrigation pipe

Fig. 13. Description of the Scenario “Fertilize with pipe”

Scenario: Fertilize using the spraying backpack
Goal: add nutrient to the plant
Context: the cistern does not have enough water to activate the irrigation pipe
Actor: Farmer
Resource: water, backpack, minerals, chart to calculate the amount minerals.
Episodes:
The farmer calculates the amount of minerals
The farmer dilutes the minerals in water
The farmer pours the mixture into the backup
The farmer sprays the liquid to the plant
The farmer washes the backpack

Fig. 14. Description of the Scenario “Fertilize with backpack”

3.4 Specification Cases Description

The description of the Specification Cases consists in combining different scenarios that share the same goal in one Specification Case. Every scenario should be categorized as one type: happy path, alternative path or exceptional path. And the episodes of every scenario should be used as the conversation of the Specification Case. Never-

theless, the conversation should be adjusted because it is necessary to determine the scope of the software system. That is, the scenarios describe the application domain, so it is necessary to identify the functionality that will be included in the software system. Thus, some actors can change (because some responsibility is performed by the software system) or even the whole episode should be adjusted. Fig. 15 describes the algorithm.

```
for each goal g
    for each scenario s_i that shares the goal g
         define the scope of software system regarding s_i
         categorize s_i as happy, alternative or exception
  path
   define a Use case u
   u.id = some common expression from all s_i
   u.goal = g
   for each s_i
         u.path = s.episodes
         Improve the description u.path
```

Fig. 15. Procedure for Specification Case description

The scenarios “fertilize using the irrigation pipe” Fig. 13 and “fertilize using the spraying backpack” Fig. 14 share the same goal “add nutrient to the plant”. So, they should be part of the same Specification Case. The happy path should be “fertilize using the irrigation pipe”, while “fertilize using the spraying backpack” is the alternative path. Then, the software system automatizes the preparation of the mixture, but the calculus about the amount of the mixture relies on the farmer. Fig. 16 describes the resulting Specification Case. Step 1 and 2 of the happy path is adapted from episode “The farmer prepares the mixture of minerals” considering the boundaries of the software system. Then, interaction 3 and 5 has the system as actor instead of the farmer as it was described in the Scenario. Finally, interaction 4 is not present in the scenario. Regarding the alternative path, the backpack is outside of the boundaries of the system and it only prepares the mixture and pours it in a container. The exceptional path was not considered in the scenarios, and it was added.

Use Case: fertilize
Goal: add nutrient to the plant
Actor: farmer
Happy path:
1. The farmer specify the amount of minerals necessaries
2. The system dilutes the minerals in water
3. The system pours the mixture into the irrigation pipe
4. The system activate the irrigation pipe
5. The system pours fresh water to wash the irrigation pipe

Alternative path:
The system does not pour the mixture into the irrigation pipe, because there is not enough water in cistern to activate the irrigation pipe.
3. The system pours the mixture into an external container.

Exceptional path:
2. The system does not have enough minerals to prepare the mixture.

Fig. 16. Description of the Use Case “Fertilize”

4 Evaluation

The framework proposed was applied to an application to manage sanitary resources related to covid-19. The system manages doctors, rooms, beds and patients. The system also manages the evolution of a patient and provides alerts according to certain workflow to follow the evolution of the patient.

Participants were 15 students of a degree course. The objective of the course is to provide a realistic experience in software development. In particular, the course emphasizes requirements practices. Nevertheless, most of the students have experience in industry since in Argentina, students generally begin to work in industry in second year of their undergraduate studies.

Participants applied the approach to describe Use Cases. The kernel sentences were provided, and the participants had to build the LEL, the Scenarios and finally Specification Cases. One of the professors of the course is a Medical Doctor, and he played the role of stakeholder and provided the information requested by the participants to enrich the description of the artifacts requested. Another professor of the course checked the quality of the Use Cases considering the knowledge they provide as well as the correct use of the template. Students also developed the software application and the professors checked the correct implementation of the functionality. Thus, we can rely on the correct specification and implementation of the functionality. And the evaluation was focused on the applicability of the approach.

The System Usability Scale (SUS) [10] [11] was used to assess the usability and applicability of the approach. Although SUS is mainly used to assess usability of software systems, it was probe to be effective to assess products and processes [7]. The System Usability Scale (SUS) consists of a 10-item questionnaire; every question must be answered in a five-options scale, ranging from ”1” (”Strongly Disagree”) to ”5” (”Strongly Agree”). Although there are 10 questions, they are related by pairs, asking the same question but in a complementary point of view in order to obtain a result of high confidence.

The calculation of the SUS score is performed in the following way. First, items 1, 3, 5, 7, and 9 are scored considering the value ranked minus 1. Then, items 2, 4, 6, 8 and 10, are scored considering 5 minus the value ranked. After that, every participant’s scores are summed up and then multiplied by 2.5 to obtain a new value ranging from 0 to 100. Finally, the average is calculated. The approach can have one of the following results: “Non acceptable” 0-64, “Acceptable” 65-84, and “Excellent” 85-100 [28]. The score obtained was 71,17. Thus, the approach can be considered as “acceptable”.

5 Related work

There are many works that describe templates to describe Use Cases, but these works do not describe a process or technique to fill the template. Cockburn et al. [13] provides an extensive guide about different templates of use cases, diagrams, and narratives with a different level of detail. They also provide best practices, but they do not provide much information about a process. The best practices they suggest consists in linking Use Cases to other models: backward traceability to the business model, for-

ward traceability to GUI, artifacts of design, test cases, etc. Denney et al. [16] describe how to deliver quality software using Use Cases. They emphasize in the template it should be used to capture the knowledge necessary to develop quality software. Nevertheless, they do not describe any process to elicit the Use Cases. Schneider et al. [32] describe some simple steps that it should be followed to describe Use Cases, but they do not provide any detail about how to do that. Savic et al. [31] propose three different levels of abstraction: description of a Use Case as a black box, description of the behavior of the Use Case, and the GUI of the software application. Tiwari et al. [36] performed a study about assessing the effectiveness of 8 different templates. They considered different criteria, and they found that no template satisfied all criteria.

Many works propose workshops as the elicitation technique. For example Alexander et al. [1] and Bittner et al. [8] recommend some specific Use Case template, but they use regular workshops to elicit information. Some other approaches use workshops combined with another techniques. Leffingwell et al. [26] recommend having preliminary interviews to plan the workshop. Alexander et al. [2] also recommend having interviews before the workshops, and they propose the use of storytelling during the workshop. Sinha et al. [33] perform workshops to request participants to describe Use Cases in colloquial way. Then, the analyst writes the Use Cases after the workshops. Richards et al. [30] propose a technique similar to Sinha et al. [33] and they consider different points of view. Thus, they consider a reconciliation of the different points of view in structured Use Cases.

Some papers rely on collaborative group sessions based on a specific technique. Gallardo et al. [20] propose a collaborative tool that manages diagrams in order to deal with different points of view. Yang-Turner et al. [35] propose an extension of the diagrams of Use Case in order to improve the dynamic of the Requirements discovery workshop. Cruz et al. [15] also propose an iterative approach to describe Use Cases: context, high level description and more detailed description. Sinha et al. [34] propose a tool to perform an automatic inspection of the Use Cases and invest effort in improving the Uses Cases more than in the initial description.

Some papers propose very detailed approaches that need very experienced analysts. Armour et al. [6] propose a very detailed approach for describing advanced Use Case. They propose three main steps (finding, describing, and refactoring) to describe very precise and detailed Use Cases. Kulak et al. [24] also propose a very detailed approach that has the following steps: mission, vision, values, statement of work, risk analysis, prototypes, and business rules. Girotto et al. [21] propose an approach to base the description of Use Cases in Business Process Model and Notation.

Regarding light approaches, the most similar to our proposed approach, Jacobson et al. [23] introduce the concept of Use Case slices to be used in combination with concepts of agile development. These Use Case slices should be described using a classic Use Case template, nevertheless, they have similarities with User Stories. Tiwari et al. [37] propose an approach to identify Use Case names and actors using natural language processing tools. Kundi et al. [25] propose an approach based on framenet, a lexical database. They propose an iterative approach to identify frame elements, describes use cases, and find new frame elements. These strategies use very different techniques to obtain knowledge: workshop to elicit orally or documents in order to perform natural language processing tools. Our approach uses three steps of

different level of complexity (LEL, Scenarios and Use Cases). These artifacts can be described iteratively using the knowledge captured in them, as well as using any other technique the requirements engineer consider in order to obtain knowledge from other sources. Thus, our proposed approach provides a framework to deal with the complexity of the knowledge while it can be integrated with other techniques familiar for the requirements engineer.

6 Conclusions

Use Cases are one of the most used artifacts to describe requirements in classic software development projects where there are a huge amount of requirements and business rules. There are many works that propose many templates to organize the knowledge in the Use Cases and there are also many proposals to evaluate que quality of the Use Cases.

Nevertheless, the responsibility and the effort of understanding the knowledge, organizing the different points of view of the stakeholders, and defining the scope of the software system rely on software developers, their skills and experience. We have proposed an approach that tackles the problems of context information and limits of the system. As such, it helps software developers, guiding the knowledge acquisition from the vocabulary (LEL) and the situations in the given environment (Scenarios). As such, producing more robust specifications.

The approach begins with simple expressions (kernel sentences). Then, they are used to describe a more complex artifact (LEL), which is the basis of another artifact that describes the behavior in the domain, the Scenarios. Then, the Scenarios are transformed in Specification Cases.

Since we rely on kernel sentences, which are focused on functional behavior, our current Specification Cases are function oriented. Future work will explore the concept of restrictions in Scenarios [17] to bring quality aspects (Non-Functional Requirements) to the Specification Cases.

This approach has different steps of elicitation and description of artifacts before the production of the specification. We have evaluated our proposed approach with a group of students, with a wide range of experience in software development and the results where positive. Nevertheless, we plan to perform a further case study where we will compare the results of our proposed approach versus the experience of writing Use Cases directly from the stakeholder. The proposed approach is the result of many years of experience, where problems of directly describing Use Cases have been found, but we still need more data to assess the effort of our approach, as well as the quality of the resulting products.

7 References

1. Alexander, Ian; Maiden, Neil. Scenarios, Stories, Use Cases. Wiley (2004).
2. Alexander, Ian; Beus-Dukic, Ljerka. Discovering Requirements: How to Specify Products and Services. Wiley (2009).

3. Antonelli, L., Rossi G., Leite, J.C.S.P., Oliveros, A: “Deriving requirements specifications from the application domain language captured by Language Extended Lexicon”, Workshop in Requirements Engineering (WER), Buenos Aires, Argentina, April 24 – 27 (2012).
4. Antonelli, L., Rossi G., Leite, J.C.S.P., Oliveros, A: “Buenas prácticas en la especificación del dominio de una aplicación”, Workshop in Requirements Engineering (WER), Montevideo, Uruguay, April 8 – 10 (2013).
5. Antonelli, L, Rossi, G, Oliveros, A: “A Collaborative Approach to Describe the Domain Language through the Language Extended Lexicon”, Journal of Object Technology, Volume 15, Number 3, issn 1660-1769, doi 10.5381/jot.2016.15.3.a3, PP 1:27 (2016).
6. Armour, Frank, Granville Miller. Advanced Use Case Modeling: Software Systems. Addison-Wesley, (2000).
7. Bangor, A., Kortum, P. T., Miller, J. T. "An empirical evaluation of the system usability scale." Intl. Journal of Human–Computer Interaction 24.6, pp. 574-594 (2008).
8. Bittner, Kurt; Spence, Ian: Use Case Modeling, Addison-Wesley Professional, 20 August (2002).
9. Boyd, Nikolas S. “Using Natural Language in Software Development.” In: Journal of Object-Oriented Programming - Report on Object Analysis and Design, 11-9 (1999)
10. Brooke, J. “SUS-A quick and dirty usability scale. Usability evaluation in industry”, 189(194), 4-7, (1996).
11. Brooke, J. "SUS: a retrospective." Journal of usability studies 8.2, pp.29-40, (2013).
12. Chomsky, N., The Logical Structure of Linguistic Theory. Plenum Press, New York, (1975).
13. Cockburn, Alistair. Writing Effective Use Cases. Addison-Wesley, (2001).
14. Cohn, Mike, User Stories Applied: For Agile Software Development, Addison Wesley, 1st Edición, ISBN: 0-321-20568-5, (2004).
15. Cruz, E. F., Machado, R. J., Santos, M. Y. "On the Decomposition of Use Cases for the Refinement of Software Requirements," 2014 14th International Conference on Computational Science and Its Applications, Guimaraes, 2014, pp. 237-240, doi: 10.1109/ICCSA.2014.54. (2014)
16. Denney, Richard. Succeeding with Use Cases: Working Smart to Deliver Quality. Addison-Wesley (2005).
17. Hadad, G., Kaplan, G., Oliveros, A., Leite, J.C.S.P.: Construcción de Escenarios a partir del Léxico Extendido del Lenguaje, in Proceedings SoST, 26JAIIO, Sociedad Argentina de Informática y Comunicaciones, Buenos Aires (1997)
18. Halle B., “Business Rules Applied”, John Wiley & Sons, Inc., New York, (2002).
19. Harris, Z.S.. Co-occurrence and transformation in linguistic structure. (Linguistic Society of America) pp. 390- 457 (1957).
20. Gallardo, J., Molina, A. I., Bravo, C. Gallego, F. "A system for collaborative building of use case models: Communication analysis and experiences: Experiences of use and lessons learned from the use of the SPACE-DESIGN tool in the domain of use case diagrams," 2014 9th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), Lisbon, pp. 1-10, (2014).
21. Girotto, A. N., Santander, V. F. A., Silva, I. F. d., Céspedes, M. A. T. "Deriving use cases from BPMN models: A proposal with computational support," 2017 36th International Conference of the Chilean Computer Science Society (SCCC), Arica, pp. 1-12, doi: 10.1109/SCCC.2017.8405122 (2017).
22. Gonçalves, J. C. de A.R., Santoro, F. M., Baião, F. A. "Collaborative narratives for business rule elicitation," 2011 IEEE International Conference on Systems, Man, and Cybernetics, Anchorage, AK, pp. 1926-1931, doi: 10.1109/ICSMC.2011.6083954. (2011).

23. Jacobson Ivar, Spence I., Bittner K. Use Case 2.0: The Guide to Succeeding with Use Cases, IJI SA (2011).
24. Kulak, Daryl, Eamonn Guiney. Use cases: requirements in context. Addison-Wesley, (2012).
25. Kundi M., Chitchyan R. "Use Case Elicitation with FrameNet Frames," 2017 IEEE 25th International Requirements Engineering Conference Workshops (REW), Lisbon, pp. 224-231, doi: 10.1109/REW.2017.53 (2017).
26. Leffingwell, Dean, Widrig, Don, Managing Software Requirements: A Use Case Approach, Addison-Wesley Professional. 7 December (2012).
27. Leite, J.C.S.P., Franco, A.P.M.: A Strategy for Conceptual Model Acquisition, in Proceedings of the First IEEE International Symposium on Requirements Engineering, San Diego, California, IEEE Computer Society Press, pp 243-246 (1993)
28. McLellan, S., Muddimer, A., Peres, S. C. "The effect of experience on System Usability Scale ratings." Journal of usability studies 7.2, pp. 56-67 (2012).
29. Tony, Morgan. Business Rules and Information Systems: Aligning IT with Business Goals, Addison-Wesley Professional; 1er edición (18 Marzo 2002)
30. Richards, D., Fure, A. Aguilera, O. "An approach to visualise and reconcile use case descriptions from multiple viewpoints," Proceedings. 11th IEEE International Requirements Engineering Conference, Monterey Bay, CA, USA, 2003, pp. 373-374, doi: 10.1109/ICRE.2003.1232792 (2003)
31. Savic D. et al., "Use Case Specification at Different Levels of Abstraction," 2012 Eighth International Conference on the Quality of Information and Communications Technology, Lisbon, pp. 187-192, doi: 10.1109/QUATIC.2012.64 (2012).
32. Schneider, Geri and Winters, Jason P. Applying Use Cases 2nd Edition: A Practical Guide. Addison-Wesley (2001).
33. Sinha, A. Paradkar, A. Kumanan P. Boguraev, B. "A linguistic analysis engine for natural language use case description and its application to dependability analysis in industrial use cases," IEEE/IFIP International Conference on Dependable Systems & Networks, Lisbon, pp. 327-336, doi: 10.1109/DSN.2009.5270320 (2009).
34. Sinha, A. Sutton S. M. Paradkar, A. "Text2Test: Automated Inspection of Natural Language Use Cases," 2010 Third International Conference on Software Testing, Verification and Validation, Paris, pp. 155-164, doi: 10.1109/ICST.2010.19 (2010).
35. Yang-Turner F. Lau, L. "Extending use case diagrams to support requirements discovery," 2011 Workshop on Requirements Engineering for Systems, Services and Systems-of-Systems, Trento, pp. 32-35, doi: 10.1109/RESS.2011.6043929 (2011).
36. Tiwari, S. Gupta, A. "A Controlled Experiment to Assess the Effectiveness of Eight Use Case Templates," 2013 20th Asia-Pacific Software Engineering Conference (APSEC), Bangkok, pp. 207-214, doi: 10.1109/APSEC.2013.37 (2013).
37. Tiwari, S. Rathore, S. S. Sagar S. Mirani, Y. "Identifying Use Case Elements from Textual Specification: A Preliminary Study," 2020 IEEE 28th International Requirements Engineering Conference (RE), Zurich, Switzerland, pp. 410-411, doi: 10.1109/RE48521.2020.00059 (2020)